Optimizing Data Transformations for Classification Tasks

José M. Valls and Ricardo Aler[**]

Universidad Carlos III de Madrid, Spain
jvalls@inf.uc3m.es
aler@inf.uc3m.es
http://www.evannai.inf.uc3m.es

Abstract. Many classification algorithms use the concept of distance or similarity between patterns. Previous work has shown that it is advantageous to optimize general Euclidean distances (GED). In this paper, data transformations are optimized instead. This is equivalent to searching for GEDs, but can be applied to any learning algorithm, even if it does not use distances explicitly. Two optimization techniques have been used: a simple Local Search (LS) and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES). CMA-ES is an advanced evolutionary method for optimization in difficult continuous domains. Both diagonal and complete matrices have been considered. Results show that in general, complete matrices found by CMA-ES either outperform or match both Local Search, and the classifier working on the original untransformed data.

Key words: Data transformations, General Euclidean Distances, Evolutionary Computation, Evolutionary-based Machine Learning

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 [1] or Nearest Neighbor classifiers [2]. It has been acknowledged that the Euclidean distance is not always the most appropriate for a particular domain, and for that reason other distances, like Mahalanobis or Chebyshev, have been proposed. For instance, the Mahalanobis distance normalizes attributes by taking into account variances and removing correlations. This distance is computed according to Eq. 1.

$$d_{ij} = [(\mathbf{x_i} - \mathbf{x_j})^T \mathbf{S}^{-1} (\mathbf{x_i} - \mathbf{x_j})]^{1/2} = [(\mathbf{x_i} - \mathbf{x_j})^T \mathbf{M}^T \mathbf{M} (\mathbf{x_i} - \mathbf{x_j})]^{1/2} \quad (1)$$

Where d_{ij} is the Mahalanobis distance between vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, S is the variance-covariance matrix of all vectors in the data set and $\mathbf{M}$ is the so-called

[**] Work supported by project TIN2008-06491-C04-04 funded by Spanish MICINN.

Mahalanobis matrix[3–5]. However, the Mahalanobis distance is computed in an unsupervised way, without taking into account the class or the training error of the classifier. In other words, the distance is not optimized for the learning algorithm or the classification task.

There are some works that optimize the distance in a supervised way. In [7], semidefinite programming has been used to learn a Mahalanobis distance for KNN. The latter work contains also a good review of previous work on metric learning for KNN. In [10], Genetic Algorithms have been used to evolve Generalized Euclidean Distances (GED) for Radial Basis Neural Networks (RBNN). GEDs look like Eq. 1, except that matrix $\mathbf{S}^{-1}$ (or matrix $\mathbf{M}$) is not computed from the data, but optimized by the Genetic Algorithm, where the fitness function is the classifier error on a training dataset.

However, this approach can only be used in classification algorithms that explicitly use distances, like RBNN, but not others like C4.5. However, a little algebra (Eq. 2) shows that the GED is equivalent to computing an Euclidean distance on a projected dataset, where the new patterns in the projected space are linear transformations of the original data: $\mathbf{x'} = \mathbf{Mx}$.

$$d_{ij} = [(\mathbf{Mx_i} - \mathbf{Mx_j})^T(\mathbf{Mx_i} - \mathbf{Mx_j})]^{1/2} \tag{2}$$

Therefore, in this paper a search method will be used to look for a transformation $\mathbf{M}$ (instead of a GED), that optimizes the training error of a learning algorithm. Any learning algorithm can be used but in the present work, a nearest neighbor algorithm (KNN) has been applied.

Two optimization techniques will be applied. First, a simple local search (LS) method is proposed as a baseline to compare with more advanced optimization methods. The second technique is CMA-ES (Covariance Matrix Adaptation Evolution Strategy), one of the best evolutionary techniques for continuous optimization in difficult non-linear domains [8][9]. An Evolution Strategy (ES) is a stochastic iterative optimization method that starts from a random population of candidate solutions, mutates them by adding Gaussian-distributed values, and selects the best performing ones. This procedure iterates until some stopping criterion is fulfilled. CMA-ES is an ES where the direction of the mutation and the step size are controlled by a covariance matrix, which is updated during the search process, so that previously realized successful steps happen again more likely. This self-adaptation of the step-size is important for our work, because a fixed step-size might be too small at the beginning of the search (where a quick exploration of the search space is required) or too large at the end, where only very small steps will get closer to the solution.

The structure of this article is as follows. Section 2 describes the method, Section 3 describes the synthetic and real domains that have been used to test the approach, and also reports the results of the experiments. Finally, Section 4 draws some conclusions and points to future work.

2 Description of the method

In this paper two optimization algorithms for finding a linear transformation of a dataset have been applied. They both attempt to minimize the classification error of a learning technique. The first one is a simple local search method that is used only as a baseline to compare with a second more advanced method: CMA-ES. In both methods, transformations are represented as matrices, which are coded directly as lists of real numbers. CMA-ES is extensively described in [8] [9]. The Matlab code available at `http://www.lri.fr/~hansen/cmaes_inmatlab.html` has been used.

With respect to the local search method, Algorithm 1 provides a summary of the algorithm. First, matrix M is initialized to the identity matrix I (line 1). In fact this means that the starting matrix m corresponds to the Euclidean distance (Matrix I). Then, the training dataset T is transformed by matrix M (line 2) and the error E_M of the learning algorithm L on T is computed (line 3). In order to prevent overfitting, E_M has been computed by means of 10-fold crossvalidation. Then, the main loop is entered (line 4) where M is mutated (line 5) and the error of M' is computed (line 6). If M' is more accurate than M, then M' is kept (line 8).

Algorithm 1: Local Search

```
 1 $M = I$
 2 $T_M = M * T$
 3 $E_M = \textbf{error}(L, T_M)$
 4 while not stopping condition do
 5     $M' = \textbf{mutate}(M)$
 6     $E_{M'} = \textbf{error}(L, T_{M'})$
 7     if $E_{M'} <= E_M$ then
 8         $M \leftarrow M'$
 9         $E_M = E_{M'}$
10     end
11 end
```

Two types of matrices have been considered: diagonal matrices (where all elements outside the diagonal are zero) and arbitrary non-diagonal matrices ($n \times n$ square matrices) that will be named 'complete matrices' in this paper. Using diagonal matrices amounts to just a weighting of the attributes by the corresponding element in the diagonal. The reason for testing these two types of structures is to check whether complete matrices are actually useful beyond a mere attribute weighting. In other words, complete matrices involve fitting many parameters ($n \times n$) and it is important to know whether they improve accuracy or rather produce overfitting.

Mutation in local search is carried out by adding a random sample from a Gaussian $N(0, 1)$ distribution. The probability of mutating a matrix element has been set to $p_m = \frac{1}{\sqrt{n \times n}}$, where n is the number of attributes and $n \times n$ is the size (the number of elements) of the complete square matrix. This means that for a complete matrix, the average number of elements that will be mutated is n. This setting worked well in preliminary experiments. One of the main differences of the simple local search algorithm and CMA-ES is that in the latter, the mutation step is controlled by a multivariate Gaussian distribution where the covariance matrix is adjusted during the course of the search, allowing for a finely tuned optimization at the end of the optimization process. In the experimental section, it will be determined to what extent this feature contributes to a high accuracy in the classification task.

CMA-ES also starts from the identity matrix and its initial step size is 1.0 (although CMA-ES will adjust this value in the course of the search). Both CMA-ES and local search stop after the same maximum number of matrix evaluations, which is determined experimentally for every domain.

Any learning algorithm L could be used. In this paper, the neighborhood-based algorithm KNN (with $k = 1$) has been selected. KNN makes very intuitive to think about what data transformations could be useful for this method. This has provided some guidance for proposing several of the synthetic domains that will be presented in the next section.

3 Experiments

In this Section, experiments carried out on several domains will be reported.

3.1 Domains

Four synthetic domains and three real world domains have been used. All of them correspond to classification problems and have numerical attributes. They are described next.

Artificial data domains Four synthetic domains have been tested: the well-known **Ripley** [6] dataset, which has been widely used in the Machine Learning literature, and three more artificial domains that we have called **RandomAttr**, **Straight0** and **Straight45**, specifically designed to check if the approach works properly.

The **Ripley** domain is a two-class problem with data distributed according to four overlapping Gaussians.

RandomAttr 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 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 ar regular intervals in a straight line passing through the origin $(0, 0)$ with an angle of 45 degrees respect to the horizontal axe. The distance between two consecutive points is 1. 100 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. Fig. 1 (left) shows a graphical representation of a subset of the domain.

Straight-0 is very similar to **Straight45**. The only difference is that all the points have been rotated 45 degrees anti-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 Fig. 1 (right) it can be seen the representation of a subset of points. In short, **Straight45** requires both rotation and scaling (a complete matrix) whereas **Straight-0** requires scaling only (a diagonal matrix).

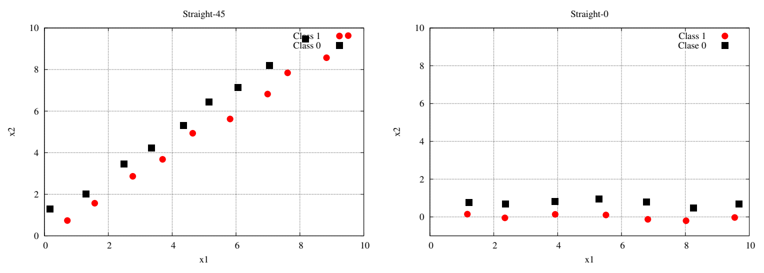

Fig. 1. Subsets of the Straight-45 (left) and Straight-0 (right) domains

Real world data domains The **Iris** (150 instances, 4 attributes, and 3 classes), **Car** (1728, 6, 4), **Bupa** (345, 7, 2), and **Wine** (178, 13, 3) domains from the UCI Machine Learning Repository have been used [1].

3.2 Experimental Results. Evolving linear transformations for KNN

This subsection reports the experimental results obtained by our method when linear transformations are evolved and KNN (with k=1) is used as classifier in the transformed space. As explained in Section 2, two optimization techniques have been used: a basic Local Search method (LS) and CMA-ES.

A 10-fold crossvalidation has been used in all cases for testing. The parameters for the search methods have been chosen in the following way: for CMA-ES, the initial standard deviation has been set to 1.0 for all the experiments. For LS, the standard deviation for the Gaussian mutation of an element is also 1.0

[1] `http://archive.ics.uci.edu/ml/`

in all cases. The maximum number of fitness evaluations has been set to the same value for both search methods by means of preliminary experiments for every domain. It is considered that the training error rate has converged when it does not decrease by more than 10^{-3} in two successive iterations. For LS, the probability of mutating a matrix element is $p_m = \frac{1}{\sqrt{n \times n}}$, as explained in Section 2. This probability is the same for all the experiments. The rest of CMA-ES parameters are set to their default values.

In all domains, the classification results in five situations are compared: for each fold, KNN classifies the original data, the data transformed by a diagonal matrix optimized by CMA-ES, by a diagonal matrix optimized by LS, by a complete matrix optimized by CMA-ES and finally, by a complete matrix optimized by LS. In all cases linear transformations are done by means of square matrices and thus, the dimension of the transformed spaces remain unaltered. Using a diagonal matrix is equivalent to scaling the original data coordinates. If a complete matrix is used there are no restrictions, being the linear transformation completely general.

Table 1 shows the classification accuracy rates obtained for all the domains and their standard deviations. The results correspond to the mean of the accuracy rates obtained in a 10-fold crossvalidation procedure.

Table 1. Classification Rate (percentage) with KNN (k=1) for the original data set and for the transformed data when diagonal and complete matrices are used. Matrices have been optimized by LS and CMA-ES

	Original Data	CMA-ES Diagonal	LS Diagonal	CMA-ES Complete	LS Complete
Straight-0	6.50 ± 5.29	$\mathbf{100.00 \pm 0.00}$	$\mathbf{100.00 \pm 0.00}$	99.50 ± 1.58	99.50 ± 1.58
Straight-45	8.50 ± 6.26	8.00 ± 6.32	7.50 ± 5.40	$\mathbf{98.50 \pm 3.37}$	$\mathbf{98.50 \pm 3.37}$
RandomAttr	49.00 ± 8.47	$\mathbf{93.67 \pm 4.83}$	81.67 ± 13.63	80.67 ± 19.10	54.67 ± 15.33
Ripley	88.64 ± 4.04	$\mathbf{89.12 \pm 4.13}$	$\mathbf{89.12 \pm 4.35}$	87.68 ± 2.19	87.44 ± 1.71
Car	87.85 ± 2.28	96.01 ± 0.84	95.60 ± 1.06	$\mathbf{97.40 \pm 1.28}$	96.64 ± 1.75
Iris	96.00 ± 3.44	93.33 ± 5.44	92.67 ± 7.34	$\mathbf{98.00 \pm 3.22}$	94.67 ± 6.88
Bupa	60.58 ± 9.52	60.58 ± 8.49	61.74 ± 5.35	$\mathbf{65.22 \pm 8.76}$	64.93 ± 8.31
Wine	78.27 ± 10.24	92.67 ± 4.55	$\mathbf{94.37 \pm 4.63}$	84.83 ± 10.5	77.53 ± 9.7

In the **Straight0** domain, results show that KNN only obtains a classification rate of 6.5% on the original data set (as expected), but if the data is transformed by a diagonal matrix the rate is 100% (99.5% for a complete matrix). The slightly worse result obtained with the complete matrix is explained because the number of parameters to adjust is larger.

In the **Straight45** domain, the second row of Table 1 shows that KNN obtains a very bad classification accuracy on the original data, as expected (8.5%). As explained in the description of the domain, a diagonal matrix is not enough to obtain an adequate transformation of data (around 8%, independently of the optimization method used). On the contrary, when a complete matrix is used, the results are near to 100%.

The **RandomAttr** domain has two irrelevant attributes whose numeric range is much bigger than the relevant attributes range. That is the reason why the accuracy of KNN on the original data is 49%. Scaling the irrelevant attributes should be enough to attain a much better accuracy, thus both diagonal and complete matrices should be appropriate. The results show that CMA-ES is able to find diagonal matrices that obtain a 93.7% classification accuracy. LS performs worse in this case (81.7%). When complete matrices are used, CMA-ES attains a 80.7% accuracy rate and LS only attains a 54.7% In this domain complete matrices do not perform as well as diagonal matrices because the number of elements to adjust is bigger and only a scaling of the coordinates is neccesary. However, CMA-ES is able to find appropriate matrices whereas LS is not. In Table 1 it can also be seen that the variability of the 10 folds results is much bigger in this domain when the complete matrix is used.

Regarding to the remaining domains, in the **Ripley** domain the results with the transformed data are quite similar to the rates obtained with the original set. That means that the system can not find any linear transformation than improves the accuracy. In the **Iris** data set, diagonal matrices worsen the accuracy. However, complete matrices optimized by CMA-ES improve the results slightly. The classifier attains very good results (97.4%) with the **Car** data set when complete matrices are used. With diagonal matrices, the results (96%) are better than the corresponding to the original data (87.8%). In the **Bupa** domain the accuracy is improved when using complete matrices. KNN obtains a classification accuracy of 60% on the original data and 65% on the transformed data with complete matrices. However, the accuracy does not improve when diagonal matrices are used. Finally, in the **Wine** domain it is diagonal matrices that improve on KNN results, but in this case LS outperforms slightly CMA-ES.

Summarizing the results, it can be observed that complete CMA-ES and LS outperform KNN on the original data for nearly all domains (except Ripley and Iris, where results are very close). In general, complete CMA-ES outperforms complete LS, although the differences are not large (except in RandomAttr and Wine where both algorithms tend to fall into local minima in some of the executions). Also, in general, the complete matrix approach outperforms the diagonal one or gets similar results, except (again) in RandomAttr and Wine. This means that in some cases, both scaling and rotations are required. And in those cases where results are similar, the complete matrix approach manages to produce an appropriate result even though there are many more parameters to fit. RandomAttr and Wine are special cases where complete CMA-ES (and LS) seem to get stuck in local minima: complete matrices could in principle match the accuracy obtained by diagonal ones (because complete matrices include diagonal ones), but they do not. In the future, methods for avoiding this situation will be proposed.

4 Conclusions

This work reports on two optimization algorithms (Local Search and CMA-ES) for finding linear transformations (represented by matrices) for datasets, in order to improve the accuracy on classification tasks. The goal of the search is to find matrices that minimize the classification error on the training data. Both diagonal and complete matrices have been considered. The non-linear classifier KNN has been applied on the transformed data. The method has been tested with seven different domains, both synthetic and real, and the results show that, in general, complete matrices found by CMA-ES either outperform or match both Local Search and the classifier on untransformed data.

In this paper, KNN has been used as the base algorithms, but because our method involves domain independent optimization methods, the approach can be applied to any other classifier. In the future, we will test if other learning methods benefit from data transformations. Also, by using rectangular matrices instead of square ones, the performance of the method as a dimensionality reduction technique could be tested (in a similar vein to [11]). Finally, non-standard CMA-ES features, like re-starts, might be of use to solve the local minima issues reported in this paper.

References

1. J.E. Moody and C. Darken. Fast Learning in Networks of Locally Tuned Processing Units. *Neural Computation*, 1:281–294, 1989.
2. T.M. Cover and P.E. Hart. Nearest Neighbor Pattern Classification. *IEEE Trans. Inform. Theory*, 13(1):21–27,1967.
3. C.G.Atkenson, A.W.Moore, and S.Schaal. Locally Weighted Learning. *Artificial Intelligence Review*, 11:11–73, 1997.
4. J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles.* Addison-Wesley, 1974.
5. S. Weisberg. *Applied Linear Regression.* New York: John Wiley and Sons, 1985.
6. B.D. Ripley. Pattern Recognition and Neural Networks *Cambridge: Cambridge University Press*, 1996.
7. K.Q. Weinberger et al. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Neural Information Processing Systems.* 2005
8. N. Hansen and A. Ostermeier. Completely Derandomized Self-adaptation in Evolution Strategies. *Evolutionary Computation* 9(2):159–195. 2001.
9. A. Ostermeier and A. Gawelczyk Nikolaus Hansen. A Derandomized Approach to Self-Adaptation of Evolution Strategies. *Evolutionary Computation.* 4(2):369-380. 1994.
10. J.M. Valls and R. Aler and O. Fernández. Evolving Generalized Euclidean Distances for Training RBNN. *Computing and Informatics.* 26:33-43. 2007.
11. A. Sierra, and A. Echeverra. Evolutionary Discriminant Analysis. *IEEE Transactions on Evolutionary Computation,* vol. 10 (1), 81-92, 2006