Approximate Linear Dependence as a Design Method for Kernel Prototype-based Classifiers

David N. Coelho[1] and Guilherme A. Barreto[2]

[1] Federal University of Ceará, Campus of Sobral, Ceará, Brazil
davidcoelho89@gmail.com,

[2] Department of Teleinformatics Engineering, Federal University of Ceará
Center of Technology, Campus of Pici, Fortaleza, Ceará, Brazil
gbarreto@ufc.br

Abstract. The approximate linear dependence (ALD) method is a sparsification procedure used to build a dictionary of samples extracted from a data set. The extracted samples are approximately linearly independent in a high-dimensional kernel reproducing Hilbert space. In this paper, we argue that the ALD method itself can be used to select relevant prototypes from a training data set and use them to classify new samples using kernelized distances. The results obtained from intensive experimentation with several datasets indicate that the proposed approach is viable to be used as a standalone classifier.

Keywords: Prototype-based classifiers, sparsification, approximate linear dependence, kernel classifiers, kernel SOM.

1 Introduction

Kernel-based methods have been introduced with the aim of developing nonlinear versions of linear supervised or unsupervised machine learning algorithms [7]. The underlying idea is to apply a kernel function $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ to any pair of training vectors so that the result can be interpreted as an inner product of a mapping function $\phi(\mathbf{x})$, where $\phi : \mathcal{X} \rightarrow F$, and F is a high-dimensional reproducing kernel Hilbert space (RKHS) (a.k.a. the feature space) [16]: $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. It should be noted that the nonlinear feature mapping $\phi(\cdot)$ is usually unknown. Thus, by means of the kernel function, inner products in the feature space are computed implicitly, i.e. without using the feature vectors directly. This appealing property of kernel methods has then been referred to as the *kernel trick*.

The process of *kernelization* has also been applied to prototype-based algorithms, such as the K-means [12], the self-organizing map (SOM) [10], the neural gas (NG) network [13] and the learning vector quantization (LVQ) [9], producing their kernelized versions: the kernel K-means [17], the kernel SOM (KSOM) [11], the kernel NG (KNG) [14] and the kernel LVQ (KLVQ) [6].

The performances of standard and kernelized versions of prototype-based classifiers are highly dependent on the number of labeled prototypes. Although

it is possible to make the set of prototypes either adaptive [1] or optimally determined by means of evolutionary algorithms [15], in the vast majority of the applications that number is set by trial and error or exhaustive grid search.

Bearing this issue in mind, in this paper we develop a simple design scheme for building kernelized prototype-based classifiers by means of the approximate linear dependence (ALD) method, which is a sparsification procedure widely used in the field of kernel adaptive filtering [5]. The proposed approach automatically selects a dictionary of samples extracted from the original data set. The dictionary, the size of which is a function of a single scalar parameter, is then used to classify a new sample using a kernelized nearest neighbor scheme. A set of experiments with benchmarking data sets confirm the viability of the proposed approach.

The remainder of this paper is organized as follows. In section 2, the fundamentals of prototype-based classification and some kernel-based methods are briefly reviewed. In section 3, the approximate linear dependence method and the proposed framework is developed. The simulation results are reported and discussed in Section 4. The paper is concluded in Section 5.

2 Basics of Prototype-Based Classification

Prototype based algorithms, as SOM and LVQ, learn from samples $\{(\mathbf{x}_i, y_i) \in \mathbb{R}^p \times \{1, ..., C\} \,|i = 1, ..., N\}$ a mapping (projection) from a high-dimensional continuous input space $\mathcal{X}$ onto a low-dimensional discrete space $\mathcal{A}$ of Q neurons. The map $i^*(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{A}$, defined by the weight matrix $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_Q\}, \mathbf{w}_i \in \mathbb{R}^p \subset \mathcal{X}$, assigns to each input vector $\mathbf{x}(n) \in \mathbb{R}^p \subset \mathcal{X}$ a winning prototype $i^*(n) \in \mathcal{A}$, determined by

$$i^*(n) = \arg\min_{\forall i} \|\mathbf{x}(n) - \mathbf{w}_i(n)\|^2, \tag{1}$$

where $\|\cdot\|$ denotes the Euclidean distance and n symbolizes a discrete time step associated with the iterations of the algorithm. This function can be kernelized as

$$\begin{aligned} i^*(n) &= \arg\min_{\forall i} \|\phi(\mathbf{x}(n)) - \phi(\mathbf{w}_i(n))\|^2, \\ &= \arg\min_{\forall i} J_i(\mathbf{x}(n)), \end{aligned} \tag{2}$$

where $J_i(\mathbf{x}(n))$ can be defined as

$$\begin{aligned} J_i(\mathbf{x}(n)) &= \|\phi(\mathbf{x}(n)) - \phi(\mathbf{w}_i(n))\|^2, \\ &= (\phi(\mathbf{x}(n)) - \phi(\mathbf{w}_i(n)))^T(\phi(\mathbf{x}(n)) - \phi(\mathbf{w}_i(n))), \\ &= \phi(\mathbf{x}(n))^T\phi(\mathbf{x}(n)) + \phi(\mathbf{w}_i(n))^T\phi(\mathbf{w}_i(n)) - 2\phi(\mathbf{x}(n))^T\phi(\mathbf{w}_i(n)), \\ &= k(\mathbf{x}(n), \mathbf{x}(n)) + k(\mathbf{w}_i(n), \mathbf{w}_i(n)) - 2k(\mathbf{x}(n), \mathbf{w}_i(n)). \end{aligned} \tag{3}$$

The prototype-based algorithms are distinguished by the update rules of their weight matrices. With the SOM algorithm, all the prototypes are updated by

the following rule

$$\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \eta(n)h(i^*, i; n)[\mathbf{x}(n) - \mathbf{w}_i(n)] \tag{4}$$

where $0 < \eta(n) < 1$ is the learning rate and $h(i^*, i; n)$ is a weighting function which limits the neighborhood of the winning neuron. On the other hand, in LVQ1, just the winner prototype is update by the following rule

$$\mathbf{w}_{i^*}(n+1) = \begin{cases} \mathbf{w}_{i^*}(n) + \eta(n)\left[\mathbf{x}(n) - \mathbf{w}_{i^*}(n)\right], y(\mathbf{x}(n)) = y(\mathbf{w}_{i^*}(n)) \\ \mathbf{w}_{i^*}(n) - \eta(n)\left[\mathbf{x}(n) - \mathbf{w}_{i^*}(n)\right], y(\mathbf{x}(n)) \neq y(\mathbf{w}_{i^*}(n)) \end{cases} \tag{5}$$

where $y(\mathbf{x}(n))$ and $y(\mathbf{w}_{i^*}(n))$ are the labels of the sample and the winner prototype respectively.

These updating rules can be also kernelized. In the Energy Function Kernel SOM (EF-KSOM), for example, the update rule is defined as

$$\mathbf{w}_i\left(n+1\right) = \mathbf{w}_i\left(n\right) + \eta\left(n\right) h\left(i^*, i, n\right) \nabla J_i(\mathbf{x}(n)), \tag{6}$$

where the gradient vector $\nabla J_i(\mathbf{x}(n))$ is defined as $\nabla J_i(\mathbf{x}(n)) = \frac{\partial J_i(\mathbf{x}(n))}{\partial \mathbf{w}_i(n)}$.

In the next subsection, some functions that can be used as kernels are reviewed.

2.1 Kernel Functions

The linear Kernel is the simplest one, where this function's output is equal to the dot product of two input vectors. This kernel, for two given vectors, $\mathbf{x} \in \mathbb{R}^p$ and $\mathbf{y} \in \mathbb{R}^p$, can be formally defined as

$$k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}. \tag{7}$$

The Gaussian kernel function has the following general form:

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\gamma^2}\right), \tag{8}$$

where $\gamma > 0$ is a scale parameter (a.k.a. the width parameter, in the current context). A suitable value for the hyperparameter γ should be carefully tuned to the problem at hand [4]. If it is overestimated, the exponential behaves almost linearly and the projection to high-dimensional feature space loses its nonlinear character. If it is underestimated, the function will lack regularization and decision boundaries tend to become highly sensitive to noise in training data.

The Cauchy kernel function has the following general form:

$$k(\mathbf{x}, \mathbf{y}) = \left(1 + \frac{\|\mathbf{x} - \mathbf{y}\|^2}{\gamma^2}\right)^{-1}, \tag{9}$$

where $\gamma > 0$ is a scale parameter.

This kernel function is a long-tailed kernel, a term borrowed from Probability for denoting distributions in which too small or too large values have large probability to occur, in contrast to the Gaussian distribution for which values far from the mean rarely occur. For this reason, the Cauchy kernel can be used to give long-range influence and sensitivity over the high-dimensional feature space [4].

The Log kernel function was introduced in [2] and its expression is given by

$$k(\mathbf{x}, \mathbf{y}) = -\log\left(1 + \frac{\|\mathbf{x} - \mathbf{y}\|^2}{\gamma^2}\right), \tag{10}$$

where log denotes the natural logarithm.

The Log kernel function belongs to a class of "not strictly positive definite" kernel functions, named *conditionally definite positive* kernel functions[3], which has been shown anyway to perform very well in many practical applications.

In the next section, the ALD method is briefly described and the proposed framework is shown.

3 The Proposed Approach

As mentioned, the training method to be proposed is based on the ALD criterion [5], which is a sparsification procedure for the construction of a dictionary consisting of a subset of the training samples $\mathcal{D}_{t-1} = \{\tilde{\mathbf{x}}_j\}_{j=1}^{m_{t-1}}$. The samples in $\mathcal{D}_{t-1}$ are *approximately* linearly independent feature vectors. The goal of the proposed approach is to take the samples of the dictionary as prototype vectors in feature space, so that they can be used in a kernelized nearest neighbor classification scheme.

At training time step t ($2 \leq t \leq N$), with N denoting the number of training samples, after having observed $t-1$ training samples, the dictionary $\mathcal{D}_{t-1}$ is comprised of a subset of m_{t-1} relevant training inputs $\{\tilde{\mathbf{x}}_j\}_{j=1}^{m_{t-1}}$. When a new incoming training sample $\mathbf{x}_t$ is available, one must test if it should be added or not to the dictionary. In order to do this, it is necessary to estimate a vector of coefficients $\mathbf{a} = \left(a_1, ..., a_{m_{t-1}}\right)^T$ satisfying the ALD criterion

$$\delta_t \overset{def}{=} \min_{\mathbf{a}} \left\| \sum_{j=1}^{m_{t-1}} a_j \boldsymbol{\phi}\left(\tilde{\mathbf{x}}_j\right) - \boldsymbol{\phi}\left(\mathbf{x}_t\right) \right\|^2 \leq \nu, \tag{11}$$

where ν is the sparsity level parameter. Developing the minimization problem in Eq. (11) and using $\kappa(\mathbf{x}, \mathbf{y}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{y}) \rangle$, we can write

$$\delta_t \overset{def}{=} \min_{\mathbf{a}} \left\{ \sum_{i,j=1}^{m_{t-1}} a_i a_j \kappa\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right) - 2 \sum_{i,j=1}^{m_{t-1}} a_j \kappa\left(\tilde{\mathbf{x}}_i, \mathbf{x}_t\right) + \kappa\left(\mathbf{x}_t, \mathbf{x}_t\right) \right\}, \tag{12}$$

[3] Let $\mathcal{X}$ be a nonempty set. A kernel $k(\cdot, \cdot)$ is called *conditionally positive definite* if and only if it is symmetric and $\sum_{j,k}^{n} c_j c_k k(\mathbf{x}_j, \mathbf{x}_k) \geq 0$, for $n \geq 1$, $c_1, \ldots, c_n \in \mathbb{R}$ with $\sum_{j=1}^{n} c_j = 0$ and $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$.

or, using the matrix notation,

$$\delta_t = \min_a \left\{ \mathbf{a}^T \tilde{\mathbf{K}}_{t-1} \mathbf{a} - 2\mathbf{a}^T \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t) + k_{tt} \right\}, \tag{13}$$

where $\left[\tilde{\mathbf{K}}_{t-1}\right]_{i,j} = \kappa(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$, $\left(\tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)\right)_i = \kappa(\tilde{\mathbf{x}}_i, \mathbf{x}_t)$, and $k_{tt} = \kappa(\mathbf{x}_t, \mathbf{x}_t)$, with $i, j = 1, ..., m_{t-1}$. Solving (13) leads to the optimal $\mathbf{a}_t$, given by

$$\mathbf{a}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t), \tag{14}$$

so that the ALD condition can be rewritten as

$$\delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}(\mathbf{x}_t)^T \mathbf{a}_t \leq \nu. \tag{15}$$

If $\delta_t > \nu$, then the sample $\mathbf{x}_t$ must be added to the dictionary; that is, $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{x}_t\}$ and $m_t = m_{t-1}$. However, if $\delta_t < \nu$, the sample is approximate linear dependent and must not be added to the dictionary.

For the purpose of classification, the ALD-based selection of prototype vectors for the dictionary can be carried out in two very simple ways, which are described next.

Design Method 1 - Randomly select an initial data sample. This sample will be the first element of the dictionary. Then, take the remaining samples of the training data set, one-by-one, and apply the ALD criterion according to (14) and (15). Note that each prototype vector in $\mathcal{D}_t =$ carries its class label for the sake of classification. The classifier designed by this method will be henceforth referred to by the acronym KNN-ALD-1 (*kernel nearest neighbor via ALD criterion 1*).

Design Method 2 - According to this method we have to build one dictionary per class. For a problem with C classes, it is required C dictionaries $\mathcal{D}_t^{(k)}$, $k = 1, 2, \ldots, C$. For this purpose, apply the Design Method 1 to the data samples of the k-th class, $k = 1, 2, \ldots, C$. Repeat this procedure for all classes individually. Merge the class-conditional dictionaries into a single larger dictionary: $\mathcal{D}_t = \mathcal{D}_t^{(1)} \cup \mathcal{D}_t^{(2)} \cup \cdots \cup \mathcal{D}_t^{(C)}$. The classifier designed by this method will be henceforth referred to as the KNN-ALD-2 (*kernel nearest neighbor via ALD criterion 2*).

For the classification of a new data sample, use the kernelized distance in Eq. (2) in order to find the closest prototype. The search is executed over the samples in the dictionary. Assign to that sample, the same class of the nearest prototype.

It should be noted that the only hyperparameters of the proposed approach are ν (the sparsity level) and those associated with the chosen kernel function (as the scale parameter γ). However, since the kernel parameters are common to all kernel-based methods, the only tunable parameter of the proposed approach is the sparsity level ν.

4 Results and Discussion

In this section, we report the results of comprehensive computer simulations verifying the classification performance of the proposed algorithm, with different kernels, when applied to real-world datasets. For all the data sets, the z-score

normalization is used, so that all attributes have zero empirical mean and unit variance. Moreover, the algorithms were implemented from scratch using MATLAB's script language and the simulations were run on a HP notebook with 2.70 GHz Intel Core i7 processor, 16 GB of RAM memory and Windows 10 Home operating system.

4.1 Initial Tests

We start with the iris data set[4] to verify how the number of prototypes and the classifier accuracy change as we modify: the hyperparameters ν and γ, the kernel functions, and the dictionary build method. This data set contains 3 classes of 50 samples each, where each class refers to a type of iris plant, and each sample is a vector of 4 attributes.

The results of the proposed method, using the linear kernel are shown at table 1. As this kernel function does not have an hyperparameter, only the build method and the sparsity level ν are verified here. We can notice that bigger values of ν lead to dictionaries with less prototypes, because it becomes harder to a new sample not be considered approximate linear dependent - see Eq. (15). For small enough values of ν, all the samples from the training data set will be chosen as prototypes. Comparing lines 2 and 6 of this table, we can notice that, with almost the same number of prototypes (12 and 13), higher accuracy rates are achieved, both in the training and test data sets, by the KNN-ALD-2 classifier. It is important to mention that this quantity of prototypes corresponds to 12% of the entire training data set.

Table 1. Preliminary tests with Iris data set and linear kernel.

Method	ν	**Kernel**	γ	acc_tr	acc_ts	#prototypes	#class 1	#class 2	#class 3
KNN-ALD-1	0.001	linear	.	0.956	0.905	49	15	13	21
KNN-ALD-1	0.01	linear	.	0.867	0.849	12	3	3	6
KNN-ALD-1	0.1	linear	.	0.759	0.739	5	1	2	2
KNN-ALD-2	0.001	linear	.	0.995	0.936	89	30	30	29
KNN-ALD-2	0.01	linear	.	0.942	0.912	30	10	10	10
KNN-ALD-2	0.1	linear	.	0.907	0.890	13	4	4	5

When using the Gaussian and Cauchy kernels, as we increase the value of the scale parameter γ, the number of chosen prototypes decreases. In order to have good classification performance and few prototypes, both ν and γ should be optimized. Also, with these kernel functions, the KNN-ALD-2 classifier achived the best results. These concepts are illustrated in table 2, where some results of the Gaussian kernel are shown.

Unlike the previously mentioned kernel functions, the number of prototypes increases as the value of γ also increases. Also, the values for ν should be negative, as the maximum value of this kernel is zero - see Eq. (10).

[4] https://archive.ics.uci.edu/ml/datasets/iris

Table 2. Preliminary tests with Iris data set and Gaussian kernel.

Build Method	v	Kernel Type	γ	acc_tr	acc_ts	prototypes	class 1	class 2	class 3
1	0.001	Gaussian	20	0.987	0.931	88	30	28	30
1	0.01	Gaussian	20	0.914	0.899	21	7	5	9
1	0.1	Gaussian	2	0.949	0.915	24	7	7	10
2	0.001	Gaussian	20	1	0.932	99	33	32	34
2	0.01	Gaussian	20	0.954	0.909	32	9	10	13
2	0.1	Gaussian	2	0.957	0.917	30	8	10	12
2	0.1	Gaussian	5	0.936	0.911	18	5	5	8
2	0.1	Gaussian	10	0.911	0.882	12	3	4	5

4.2 General Tests

In the following experiments, for each evaluated data set, we test 8 variants of the proposed algorithm, consisting of 4 different kernel functions (linear, Gaussian, Cauchy, log) and 2 build methods (an unique dictionary or one dictionary per class). Also, 50 independent training-testing runs are executed. For each run, three steps are performed, namely: (*i*) holdout (partition of the data between training and test sets), (*ii*) training (hyperparameters optimization and parameters update), (*iii*) performance testing. For the holdout step, the data is randomly divided as follows: 70% for training and remaining 30% for test. At the end of test phase, several statistical figures of merit for the performance of each classifier are computed.

Finally, we perform a 5-fold cross-validation strategy in order to search the optimal values of the hyperparameters ν (from the ALD criterion) and γ (from the Gaussian, Cauchy and Log kernel functions). The figure of merit for evaluating the algorithms performance while choosing the optimal hyperparameters is given by

$$J_h = \alpha - \beta.n_p \tag{16}$$

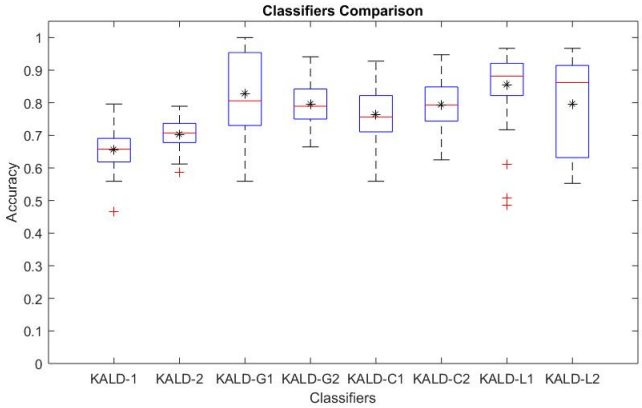
where α is the classifier accuracy, n_p is the percentage of prototypes in relation to the number of training samples, and β is a weighting between these two factors. By increasing β, this evaluation penalizes hyperparameters that lead the algorithm to build dictionaries with a large number of prototypes.

The motor failure data set was the first to be investigated. It consists of 294 feature vectors, each one containing 6 harmonics of the Fourier Transform from a line current measurement of a three phase induction motor [3]. These samples fall into 7 classes, where 1 is for normal condition (42 samples) and the other 6 are from short-circuit condition (252 samples). In [3], the best results were reached when the problem was treated as a binary one and when the classes were balanced (adding 210 artificial samples of normal condition to the data set). So, the same methodology was used at this paper.

The results for this methodology, using the data set of 504 samples (252 for each class) and $\beta = 0.5$ - see Eq. (16) - are depicted at figure 1. First, it is possible to infer that, for this data set, the classification performance is improved when the kernel functions, different from the linear one, are used. if we just analyze,

for example, the best result of the proposed approach, using the Gaussian Kernel and the dKNN-ALD-1 classifier, the following results were achieved: the classifier reached 100% of accuracy but it needed to use $62,5\%$ of the samples from the training data set (220 prototypes from 352 training samples).

Fig. 1. Results using the motor failure data set



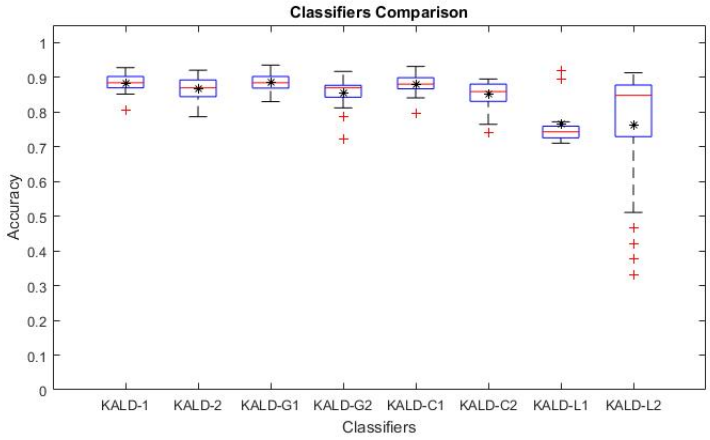
In order to test a data set with more features and samples, the Pap-smear data set [5] was used. It consists of 917 images of Pap-smear cells where each cell is described by 20 numerical features, and the cells fall into 7 classes. 3 classes are from normal cells (totaling 242 samples) and 4 classes are from abnormal cells (totaling 675 samples) [8]. This data set was also considered as a binary one. The results achieved by the proposed approach, using $\beta = 1$ are represented in figure 2. First, the worst mean and maximum accuracy rates, for this data set, were achieved by using the log kernel. Finally, if we just analyze, for example, the best result of the proposed approach, using the Gaussian Kernel and the KNN-ALD-1 classifier, the following results were achieved: the classifier reached 92% of accuracy using just $3,27\%$ of samples from the training data set (21 prototypes from 641 training samples).

5 Conclusions and Further Work

In this paper, we presented a new prototype-based classifier that uses the ALD method for selecting the relevant prototypes of a training data set and uses kernelized distances in order to classify new data. This method has the advantage of having just few hyperparameters (sparsity level ν and the kernel functions' parameter γ) to optimize, although some questions still need to be further investigated, such as the first element choice of the dictionary. In preliminary tests

[5] http://mde-lab.aegean.gr/downloads

Fig. 2. Results using the Pap-smear data set



with Iris data base, we indicate some characteristics of the hyperparameters and dictionary build methods. In the general tests, the maximum results were achieved by using the Gaussian kernel function and the KNN-ALD-1 classifier. Also in the general tests, we shown that this approach can be successfully applied to classification, as it reached 100% of maximum accuracy for the motor failure data set and 92% with the Pap-smear data set using $62,5\%$ and $3,27\%$ training samples of each data set respectively.

Currently, we are evaluating if the training part of the proposed algorithm can be used as an initialization approach to another prototype based algorithms, such as SOM and LVQ, by choosing the initial prototypes, avoiding trial-and-error methods.

Acknowledgments: This study was financed by the following Brazilian research funding agencies: CAPES (Finance Code 001) and CNPq (grant 309451/2015-9).

References

1. Albuquerque, R.F., de Oliveira, P.D., Braga, A.P.d.S.: Adaptive fuzzy learning vector quantization (AFLVQ) for time series classification. In: North American Fuzzy Information Processing Society Annual Conference, pp. 385–397. Springer (2018)
2. Boughorbel, S., Tarel, J.P., Boujemaa, N.: Conditionally positive definite kernels for SVM based image recognition. In: Proceedings of the IEEE International Conference on Multimedia & Expo (ICME'2005), pp. 1–4 (2005)
3. Coelho, D.N., Barreto, G., Medeiros, C.M., Santos, J.D.A.: Performance comparison of classifiers in the detection of short circuit incipient fault in a three-phase induction motor. In: Computational Intelligence for Engineering Solutions (CIES), 2014 IEEE Symposium on, pp. 42–48. IEEE (2014)
4. de Souza, C.R.: Kernel functions for machine learning applications. URL http://crsouza.com/2010/03/17/kernel-functions-for-machine-learning-applications/

5. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least squares algorithm. IEEE Transactions on signal processing **52**(8), 2275–2285 (2004)
6. Hofmann, D., Hammer, B.: Sparse approximations for kernel learning vector quantization. In: Proceedings of the ESANN'2013, pp. 549–554 (2013)
7. Jäkel, F., Schölkopf, B., Wichmann, F.A.: A tutorial on kernel methods for categorization. Journal of Mathematical Psychology **51**(6), 343–358 (2007)
8. Jantzen, J., Norup, J., Dounias, G., Bjerregaard, B.: Pap-smear benchmark data for pattern classification. Nature inspired Smart Information Systems (NiSIS 2005) pp. 1–9 (2005)
9. Kohonen, T.: Improved versions of learning vector quantization. In: Neural Networks, 1990., 1990 IJCNN International Joint Conference on, pp. 545–550. IEEE (1990)
10. Kohonen, T.: The self-organizing map. Proceedings of the IEEE **78**(9), 1464–1480 (1990)
11. Lau, K.W., Yin, H., Hubbard, S.: Kernel self-organising maps for classification. Neurocomputing **69**(16), 2033–2040 (2006)
12. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297. University of California Press (1967)
13. Martinetz, T.M., Berkovich, S.G., Schulten, K.J., et al.: 'neural-gas' network for vector quantization and its application to time-series prediction. IEEE transactions on neural networks **4**(4), 558–569 (1993)
14. Qinand, A., Suganthan, P.N.: Kernel neural gas algorithms with application to cluster analysis. In: null, pp. 617–620. IEEE (2004)
15. Soares Filho, L.A., Barreto, G.A.: On the efficient design of a prototype-based classifier using differential evolution. In: Differential Evolution (SDE), 2014 IEEE Symposium on, pp. 1–8. IEEE (2014)
16. Yin, H.: On the equivalence between kernel self-organising maps and self-organising mixture density networks. Neural Networks **19**(6), 780–784 (2006)
17. Zhang, R., Rudnicky, A.I.: A large scale clustering scheme for kernel k-means. In: Pattern Recognition, 2002. Proceedings. 16th International Conference on, vol. 4, pp. 289–292. IEEE (2002)