LVQ-type Classifiers for Condition Monitoring of Induction Motors: A Performance Comparison

Diego P. Sousa[1], Guilherme A. Barreto[1], Charles C. Cavalcante[1], and Cláudio M. S. Medeiros[2]

[1] Federal University of Ceará - Department of Teleinformatics Engineering
Campus of Pici, Center of Technology, Fortaleza, Ceará, Brazil
`diegoperdigao@gmail.com, {gbarreto,charles}@ufc.br`

[2] Federal Institute of Ceará, Department of Industry
Laboratory of Energy Processing, Fortaleza, Ceará, Brazil
`claudiosa@ifce.edu.br`

Abstract. In this paper, we introduce a design methodology for prototype-based classifiers, more specifically the well-known LVQ family, aiming at improving their accuracy in fault detection/classification tasks. A laboratory testbed is constructed to generate the datasets which are comprised of short-circuit faults of different impedance levels, in addition to samples of the normal functioning of the motor. The generated data samples are difficult to classify as normal or faulty ones, especially if the faults are of high impedance (usually misinterpreted as non-faulty samples). Aiming at reducing misclassification, we use K-means and cluster validation techniques for finding an adequate number of labeled prototypes and their correct initialization for the efficient design of LVQ classifiers. By means of comprehensive computer simulations, we compare the performances of several LVQ classifiers in the aforementioned engineering application, showing that the proposed methodology eventually leads to high classification rates.

Keywords: Learning vector quantization, prototype-based classifiers, fault detection, induction motors, condition monitoring

1 Introduction

The family of learning vector quantization (LVQ) algorithms is comprised of prototype-based neural network (NN) models which have been used as alternatives to more traditional approaches (e.g. MLP and RBF networks). LVQ classifiers present classification accuracies at least as high as that of any other NN algorithm [1] and are simpler to interpret due to the local nature of the prototypes, which are positioned at representative regions (Voronoi cells) of the data.

The finest feature of standard LVQ algorithms, that of interpretability due to the local nature of the prototype vectors, is also their main drawback. That is, the performance of an LVQ classifier is highly dependent on the prespecified number of labeled prototypes. Previous works have been developed trying to make the set of prototypes either adaptive [2] or optimally determined by means of evolutionary algorithms [3]. However, in the vast majority of the applications, that number is set by trial and error or exhaustive grid search (see [4, 5] for excellent surveys on LVQ-based and other prototype-based classifiers).

From the exposed, in this paper, we aim to introduce a systematic methodology for finding a suitable number of prototypes and their reliable initialization. Roughly speaking, instead of inserting and/or removing prototypes on the fly as did by adaptive LVQ classifiers, we resort to clustering strategies to find the optimum number of prototypes per class. For the sake of simplicity, we use the K-means and well-known cluster validation indices, but any other clustering methodology can be used as well.

For assessing the proposed methodology we evaluate the state of the art in LVQ models on a fault detection/classification dataset obtained from 3-phase AC induction motors. Our target task is the identification of inter-turn short-circuit faults in the stator winding, which we have been investigating lately using standard powerful nonlinear classifiers, such as the MLP and the SVM [6] and SOM techniques [7, 8]. For this purpose, we built a lab scale testbed for simulating faults of different impedance levels with different degrees of severity.

The remainder of the paper is divided as follows: in section 2, the basics of cluster validation techniques are presented; in section 3, LVQ-based classifiers are described; in section 4, the experimental data acquisition is explained; in section 5 our proposal is introduced and the results are shown and discussed; finally, in section 6, the conclusions are made.

2 Basics of Cluster Validation Techniques

Techniques for cluster validation are used *a posteriori* to evaluate the results of a given clustering algorithm. It should be noted, however, that each cluster validation technique has its own set of assumptions, so that the final results may vary across the chosen techniques.

2.1 Cluster Validity Indices

Some well-known indices available in the clustering literature are described next. We denote K as the number of clusters, K_{max} is the maximum allowed number of clusters, d as the number of features, $\bar{\mathbf{x}}$ as the centroid of the $d \times N$ data matrix $\mathbf{X}$, n_i as the number of objects in cluster C_i, $\mathbf{c}_i$ as the centroid of cluster C_i, and $\mathbf{x}_l^{(i)}$ as the l-th feature vector, $l = 1, \ldots, n_i$, belonging of the cluster C_i.
(i) The *Davies-Bouldin* (DB) index [9] is a function of the ratio of the sum of within-cluster scatter to between-cluster separation, and it uses the clusters' centroids for this purpose. Initially, we need to compute the scatter within the i-th cluster and the separation between the i-th and j-th clusters, respectively,

as

$$S_i = \left[\frac{1}{n_i}\sum_{l=1}^{n_i}\|\mathbf{x}_l^{(i)} - \mathbf{c}_i\|^2\right]^{1/2} \quad \text{and} \quad d_{ij} = \|\mathbf{c}_i - \mathbf{c}_j\| \tag{1}$$

where $\|\cdot\|$ is the Euclidean norm. Finally, the DB index is defined as

$$DB(K) = \frac{1}{K}\sum_{i=1}^{K} R_i, \quad \text{where} \quad R_i = \max_{j \neq i}\left\{\frac{S_i + S_j}{d_{ij}}\right\}. \tag{2}$$

The value of K leading to the smallest $DB(K)$ value is chosen as the optimal number of clusters.
(*ii*) The *Dunn* index [10] is represented generically by the following expression:

$$Dunn(K) = \frac{\min_{i \neq j}\{\delta(C_i, C_j)\}}{\max_{1 \leq l \leq k}\{\Delta(C_l)\}}, \tag{3}$$

where

$$\delta(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{y} \in C_j}\{d(\mathbf{x}, \mathbf{y})\}, \quad \text{and} \quad \Delta(C_i) = \max_{\mathbf{x}, \mathbf{y} \in C_i}\{d(\mathbf{x}, \mathbf{y})\}, \tag{4}$$

with $d(\cdot, \cdot)$ denoting a dissimilarity function (e.g. Euclidean distance) between vectors. Note that, while $\delta(C_i, C_j)$ is a measure of separation between clusters C_i and C_j, $\Delta(C_i)$ is a measure of the dispersion of data within the cluster C_i. The value of K resulting in the largest $Dunn(K)$ value is chosen as the optimal number of clusters.
(*iii*) The *Calinski-Harabasz* (CH) index [11] is a function defined as

$$CH(K) = \frac{trace(\mathbf{B}_K)/(K-1)}{trace(\mathbf{W}_K)/(N-K)} \tag{5}$$

where $\mathbf{B}_K = \sum_{i=1}^{K} n_i(\mathbf{c}_i - \bar{\mathbf{x}})(\mathbf{c}_i - \bar{\mathbf{x}})^T$ is the between-group scatter matrix for data partitioned into K clusters, $\mathbf{W}_K = \sum_{i=1}^{K}\sum_{l=1}^{n_i}(\mathbf{x}_l^{(i)} - \mathbf{c}_i)(\mathbf{x}_l^{(i)} - \mathbf{c}_i)^T$ is the within-group scatter matrix for data clustered into K clusters. The $trace(\cdot)$ operator computes the sum of the elements on the main diagonal of a square matrix. The value of K resulting in the largest $CH(K)$ value is chosen as the optimal number of clusters.
(*iv*) The *Silhouette* (Sil) index [12] is defined as

$$Sil(K) = \sum_{i=1}^{N} S(i)/N, \qquad S(i) = [b(i) - a(i)]/\max\{a(i), b(i)\}, \tag{6}$$

with $a(i)$ representing the average dissimilarity of the i-th feature vector to all other vectors within the same cluster (except i itself), and $b(i)$ denoting the lowest average dissimilarity of the i-th feature vector to any other cluster of which it is not a member. The silhouette can be calculated with any dissimilarity metric, such as the Euclidean or Manhattan distances. The value of K producing the largest $Sil(K)$ value is chosen as the optimal number of clusters.

3 Prototype-Based Classifiers

Let us consider a set of training input-output patterns $\{(\mathbf{x}_l, y_l)\}_{l=1}^{N}$, where $\mathbf{x}_l \in \mathbb{R}^p$ denotes the l-th input pattern and $y_l \in \mathcal{C}$ denotes its corresponding class label. Note that y_l is a discrete variable (of either numerical or nominal nature) which may assume only one out of K values in the finite set $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$.

Given a set of labeled prototype vectors $\mathbf{m}_i \in \mathbb{R}^p$, $i = 1, \ldots, M$, for all the prototype-based classifiers to be described in this section, class assignment for a new input pattern $\mathbf{x}(t)$ is based on the following decision criterion:

$$\text{Class of } \mathbf{x}(t) = \text{Class of } \mathbf{m}_c(t), \quad \text{where} \quad c = \arg \min_{i=1,\ldots,M} d(\mathbf{x}(t), \mathbf{m}_i(t)), \tag{7}$$

in which $d(\cdot,\cdot)$ denotes a dissimilarity measure specific to the extension of LVQ and c is the index of the nearest prototype among the M ones available. In the following paragraphs, we briefly described the learning rules for finding the positions of the prototypes $\mathbf{m}_i$, $i = 1, \ldots, M$ in the data space.

Minimum Distance-to-Centroid (MDC) classifier [13]: For this classifier, we have $M = K$, i.e. the number of prototypes (M) is equal to the number of classes (K). In this case, the prototype of the i-th class is computed as the centroid of class i as $\mathbf{m}_i = \frac{1}{n_i}\sum_{\mathbf{x}\in c_i} \mathbf{x}, \quad i = 1, \ldots, K$, where n_i is the number of training examples of class i.

3.1 LVQ classifiers

For the whole family of LVQ classifiers, we have $M > K$, i.e. the number of prototypes (M) is higher than the number of classes (K). As a consequence, different prototypes may share the same label.

LVQ1 [14]: Let c be defined as in Eq. (7) for a new input pattern $\mathbf{x}(t)$. Then, the prototype $\mathbf{m}_c$ is updated as follows

$$\mathbf{m}_c(t+1) = \mathbf{m}_c(t) + s(t)\alpha(t)[\mathbf{x}(t) - \mathbf{m}_c(t)], \tag{8}$$

where $s(t) = +1$ if the classification is correct, and $s(t) = -1$ if the classification is wrong, and α is the learning rate.

LVQ2.1 [14]: In this algorithm, two prototypes $\mathbf{m}_i$ and $\mathbf{m}_j$ that are the nearest neighbors to $\mathbf{x}(t)$ are now updated simultaneously. One of them ($\mathbf{m}_i$, for example) must belong to the correct class and the other to the wrong class, respectively. Thus, the learning rules of the LVQ2.1 algorithm are given by

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_i(t)], \tag{9}$$
$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) - \alpha(t)[\mathbf{x}(t) - \mathbf{m}_j(t)], \tag{10}$$

where $\mathbf{x}(t)$ must satisfy the following condition:

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s, \quad \text{where } s = \frac{1-w}{1+w}, \tag{11}$$

where d_i and d_j are the Euclidean distances of $\mathbf{x}(t)$ from $\mathbf{m}_i$ and $\mathbf{m}_j$, respectively. A relatively 'window' w width from 0.2 to 0.3 is recommended.

LVQ3 [14]: For scenarios in which $\mathbf{x}(t)$, the nearest prototypes $\mathbf{m}_i$ and $\mathbf{m}_j$ belong to the same class, the following updating rule is applicable:

$$\mathbf{m}_k(t+1) = \mathbf{m}_k(t) + \epsilon s(t)\alpha(t)[\mathbf{x}(t) - \mathbf{m}_k(t)], \tag{12}$$

for $k \in \{i, j\}$, with $\mathbf{x}$ falling into the 'window'. In a series of experiments carried out in [14], feasible values of ϵ ranging from 0.1 to 0.5 were found, relating to $w = 0.2$ or 0.3. The optimal values for ϵ seems to depend on the size of the window, being smaller for narrower windows. An important feature of the LVQ3 algorithm is that it is self-stabilizing, in the sense that the optimal placements of the prototypes do not change in continued learning.

GLVQ [15]: The cost function is defined as follows:

$$E_{GLVQ} = \sum_{i=1}^{N} \phi(\mu(\mathbf{x})), \quad \mu(\mathbf{x}) = \frac{d^+ - d^-}{d^+ + d^-}, \tag{13}$$

where $\phi(\cdot)$ is the identity function for linear GLVQ or the the logistic function for logistic GLVQ, and μ is the relative distance difference, and $d^+ = d(\mathbf{x}, \mathbf{m}^+)$ is the squared Euclidean distance of the input pattern $\mathbf{x}(t)$ to its closest prototype $\mathbf{m}^+(t)$ having the same label, and $d^- = d(\mathbf{x}, \mathbf{m}^-)$ is the squared Euclidean distance of the input pattern $\mathbf{x}(t)$ to its closest prototype $\mathbf{m}^-(t)$ having a different class label. Considering these scenarios, the following updating rules are applicable:

$$\mathbf{m}^+(t+1) = \mathbf{m}^+(t) + \alpha(t)\phi'(\mu(\mathbf{x}))[4d^-/(d^+ + d^-)^2][\mathbf{x}(t) - \mathbf{m}^+(t)], \tag{14}$$

$$\mathbf{m}^-(t+1) = \mathbf{m}^+(t) - \alpha(t)\phi'(\mu(\mathbf{x}))[4d^+/(d^+ + d^-)^2][\mathbf{x}(t) - \mathbf{m}^-(t)]. \tag{15}$$

GRLVQ [16, 17]: In this algorithm, the cost function is defined as follows:

$$E_{GRLVQ} = \sum_{i=1}^{N} \phi(\mu(\mathbf{x})), \quad \mu(\mathbf{x}) = \frac{d^+_{\boldsymbol{\lambda}} - d^-_{\boldsymbol{\lambda}}}{d^+_{\boldsymbol{\lambda}} + d^-_{\boldsymbol{\lambda}}}, \tag{16}$$

where $\phi(\cdot)$ is the logistic function, and $d^+_{\boldsymbol{\lambda}} = d(\mathbf{x}, \mathbf{m}^+)$ is the squared Euclidean weighted distance of input pattern $\mathbf{x}(t)$ to its closest prototype $\mathbf{m}^+(t)$ having the same label, and $d^-_{\boldsymbol{\lambda}} = d(\mathbf{x}, \mathbf{m}^-)$ is the squared Euclidean weighted distance of the input pattern $\mathbf{x}(t)$ to its closest prototype $\mathbf{m}^-(t)$ having a different class label. The relevance factors can be determined by the gradient descent as

$$\boldsymbol{\lambda}(t+1) = \boldsymbol{\lambda}(t) - \epsilon_\lambda(t)\phi'\left(\frac{(\mathbf{x} - \mathbf{m}^+)^2 d^-_{\boldsymbol{\lambda}} - (\mathbf{x} - \mathbf{m}^-)^2 d^+_{\boldsymbol{\lambda}}}{(d^+_{\boldsymbol{\lambda}} + d^-_{\boldsymbol{\lambda}})^2}\right), \tag{17}$$

where ϵ_λ is the gain factor, and $\lambda_a \geq 0$ for $a = 1, \ldots, p$, and numerical instabilities are avoided by applying the normalization $\sum_{a=1}^{p} \lambda_a = 1$. Finally, the updating rules are shown below:

$$\mathbf{m}^+(t+1) = \mathbf{m}^+(t) + \alpha(t)\phi'(\mu(\mathbf{x}))[4d^-_{\boldsymbol{\lambda}}/(d^+_{\boldsymbol{\lambda}} + d^-_{\boldsymbol{\lambda}})^2][\mathbf{x}(t) - \mathbf{m}^+(t)], \tag{18}$$

$$\mathbf{m}^-(t+1) = \mathbf{m}^+(t) - \alpha(t)\phi'(\mu(\mathbf{x}))[4d^+_{\boldsymbol{\lambda}}/(d^+_{\boldsymbol{\lambda}} + d^-_{\boldsymbol{\lambda}})^2][\mathbf{x}(t) - \mathbf{m}^-(t)]. \tag{19}$$

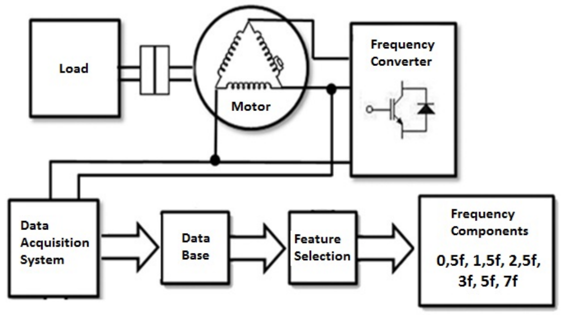


Fig. 1: Modules of the laboratory test bed and the data acquisition system.

4 Results and discussion

In this section, we evaluate the proposed methodology to find the number of prototypes and its locations for the 3 types of classes existing in the available dataset, which are represented by the labels N (normal), H (high impedance) and L (low impedance). The dataset is comprised of 294 6-dimensional labeled feature vectors, in which the attribute values represent the FFT values for the chosen 6 harmonics of the fundamental frequency of the converter drive.

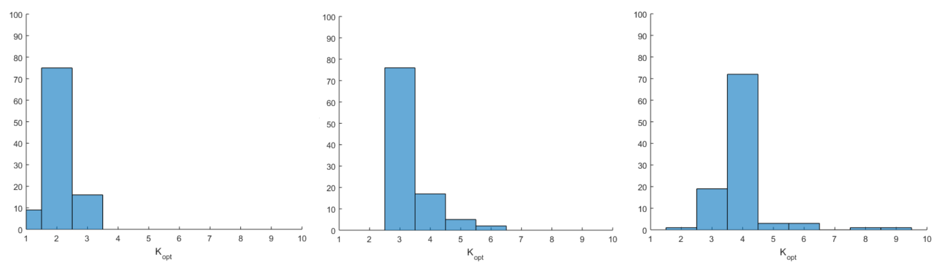
A 3-phase squirrel-cage induction motor built by WEG[3] industry is used in this study. Its main characteristics are 0.75 kW (power), 220/380 V (nominal voltage), 3.02/1.75 A (nominal current), 79.5% (efficiency), 1720 rpm (nominal rotational speed), Ip/In = 7.2 (peak to nominal current ratio), and 0.82 (power factor). The dataset is generated with this motor operating in different working conditions. The modules of the laboratory scale testbed are shown in Fig. 1, and are hereafter explained.

The task of interest can be approached either as a 3-class problem (ternary classification) or as a 2-class problem (binary classification). For the ternary classification, the distribution of samples per class is as follows: normal condition (with 42 samples), high impedance fault (with 126 samples) or low impedance fault (also with 126 samples). As a binary classification problem, we merge the samples from classes H and L and label them simply as *Faulty*. Further details on the construction of this dataset and the experimental apparatus built for its generation is given in [6–8].

For each classifier, 100 independent turns of training and testing are carried out. For each run, the four steps of the proposed methodology are executed: (*i*) the holdout (division of the data set into training and validation data sets); (*ii*) determination of the K_{opt} and prototypes' initialization via application of clustering and cluster validity techniques per data class; and, (*iv*) LVQ training and testing. At the end of each run, the accuracy rate of each classifier is determined.

The 2nd step of the methodology is comprised of two stages. In the first stage, we apply the K-means algorithm on each class individually, for $K =$

[3] `http://www.weg.net/institutional/BR/en/`



(a) N class (b) H class (c) L class

Fig. 2: Histograms of K_{opt} obtained by applying the majority voting scheme to the cluster validity techniques along 100 independent runs.

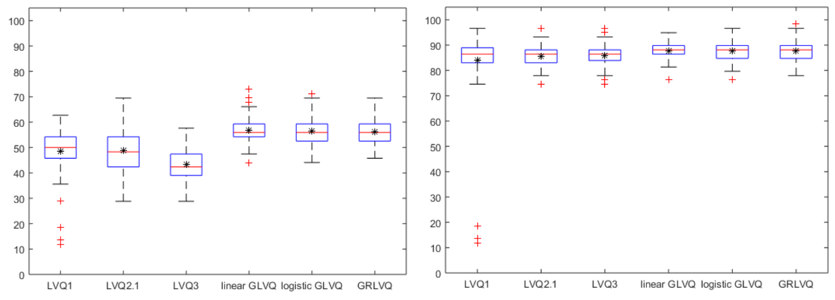
$2, 3, \cdots, K_{max} = 10$ prototypes. For each value of K, we execute 10 independent runs of the K-means algorithm and choose the set of prototypes $\{\mathbf{p}_j\}_{j=2}^{K}$ that produces the lowest MSQE[4] for each class. Using these selected sets, we compute the corresponding values of the cluster validity indices in order to choose the optimal number of prototypes per class. We use the majority voting of these cluster validation techniques to make this choice. In the 2nd stage, we initialize the LVQ classifiers using the selected K_{opt} prototypes.

The histograms of the suggested optimal number of prototypes per class resulting from the majority voting scheme along the 100 independent turns are shown in Fig. 2. By analyzing this figure, it can be seen that the setup with $K_N = 2$, $K_H = 3$, and $K_L = 4$ is the most frequent one.

The performance of each LVQ-based classifier is shown in Fig. 3a for ternary classification and in Fig. 3b for binary classification. For the binary setting, we merge the L and H classes into a single faulty class. A closer look at these figures reveals that the linear GLVQ, logistic GLVQ, and GRLVQ classifiers performed better than the other three. Then, we choose GRLVQ and the linear GLVQ (due to the smaller computational cost) for further analyses. As shown in Table 1 the accuracy metrics of these two classifiers were basically the same.

It should be pointed out that as important as maximizing the overall accuracy rate is minimizing the occurrence of certain types of errors. As observed in [6] and [8], since the classification task is unbalanced (there are 252 samples of faulty conditions and only 42 samples of normal conditions), trained algorithms are biased toward classifying normal samples as faulty ones. This means that the false alarm rate is undesirably high in this case and should be minimized, because false alarms force unexpected downtime for maintenance purposes and, hence, cause an increase in production costs.

[4] Mean squared quantization error: $MSQE = \frac{1}{n_k}\sum_{\forall \mathbf{x} \in c_k} \|\mathbf{x} - \mathbf{m}_c^k\|^2$, where n_k is the number of data samples of the class c_k and $\mathbf{m}_c^k$ is the nearest prototype belonging to class c_k.



(a) Accuracy of ternary classification. (b) Accuracy of binary classification.

Fig. 3: Boxplots of accuracy rates achieved by the evaluated LVQ classifiers.

In order to evaluate the linear GLVQ and GRLVQ in this regard, we compared their confusion matrices for the best case scenario, i.e. the one leading to the largest classification rate within the 100 independent runs. We observe in the 1st row of Tables 2 and 3 that both classifiers erroneously classified one faulty sample as a normal one (GLVQ: true L $\Rightarrow$ N, GRLVQ: true H $\Rightarrow$ N). However, only the GLVQ classified a normal sample as a faulty one (GLVQ: true N $\Rightarrow$ H) as we can see in the 1st column of those same tables. Similar behaviors are inferred for the binary task by analyzing Tables 4 and 5.

From the reported results, we can infer that the GRLVQ classifier is the best suited for the fault classification task of interest among all the evaluated LVQ variants. A nice feature of the GRLVQ is that we can check the relevance weights in order to have a clear notion which input attributes have more influence on the performance of the GRLVQ classifier. By analyzing the relevance weights' vector given by

$$\begin{aligned}\boldsymbol{\lambda} &= [\lambda_{0.5\hat{f}_c} \mid \lambda_{1.5\hat{f}_c} \mid \lambda_{2.5\hat{f}_c} \mid \lambda_{3\hat{f}_c} \mid \lambda_{5\hat{f}_c} \mid \lambda_{7\hat{f}_c}] \\ &= [0.1373|0.0952|0.1757|0.2277|0.1689|0.1952],\end{aligned} \tag{20}$$

we can see that the 4th attribute ($\lambda_{3\hat{f}_c}$) is the most relevant one, while the 2nd attribute ($\lambda_{1.5\hat{f}_c}$) is the least relevant one.

5 Conclusions and Further Work

In this paper, we introduced a clustering-based methodology for building efficient LVQ-based classifiers. Our motivation had its origin in a complex fault classification task in which we have been working now for some years. The target task of detecting inter-turn short-circuit faults is challenging (even for human experts) because of the high probability of misinterpretation of high impedance faults as normal ones. Previous experience with powerful supervised neural network based classifiers, such as the MLP and RBF networks, has challenged us to apply much

		Min	Max	Median	Std
Ternary	**GLVQ**	44.068%	72.881%	55.932%	5.575
	GRLVQ	45.763%	69.491%	55.932%	5.602
Binary	**GLVQ**	76.271%	96.610%	88.136%	3.606
	GRLVQ	77.966%	98.305%	88.136%	3.574

Table 1: Accuracy rates for the ternary and binary classification tasks achieved by linear GLVQ and GRLVQ classifiers.

Linear GLVQ		**Actual C.**		
		N	**H**	**L**
Predicted Class	**N**	2	0	1
	H	2	14	9
	L	0	13	18

Table 2: Best ternary confusion matrix (GLVQ).

GRLVQ		**Actual C.**		
		N	**H**	**L**
Predicted Class	**N**	1	1	0
	H	0	21	4
	L	0	13	19

Table 3: Best ternary confusion matrix (GRLVQ).

Linear GLVQ		**Actual Class**	
		Normal	**Faulty**
Predicted Class	**Normal**	2	1
	Faulty	2	54

Table 4: Best binary confusion matrix (linear GLVQ).

GRLVQ		**Actual Class**	
		Normal	**Faulty**
Predicted Class	**Normal**	1	1
	Faulty	0	57

Table 5: Best binary confusion matrix (GRLVQ).

simpler prototype-based classifiers and get acceptable performances on the fault classification task of interest. We succeeded in reporting high accuracy rates, comparable to those achieved in previous works of our research group.

Currently, we are investigating the performances of kernelized versions of LVQ classifiers on the same fault classification task. Our ultimate goal is to develop an embedded software application capable of monitoring induction motors in an online fashion with high accuracy rates.

Acknowledgments: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors also thank CNPq (grant 309451/2015-9) for the financial support and IFCE for the infrastructure of the Laboratory of Energy Processing.

References

1. T. Kohonen, "Improved versions of learning vector quantization," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on.* IEEE, 1990, pp. 545–550.
2. R. F. Albuquerque, P. D. de Oliveira, and A. P. d. S. Braga, "Adaptive fuzzy learning vector quantization (AFLVQ) for time series classification," in *North American Fuzzy Information Processing Society Annual Conference.* Springer, 2018, pp. 385–397.

3. L. A. Soares Filho and G. A. Barreto, "On the efficient design of a prototype-based classifier using differential evolution," in *Differential Evolution (SDE), 2014 IEEE Symposium on*. IEEE, 2014, pp. 1–8.
4. M. Biehl, B. Hammer, and T. Villmann, "Prototype-based models in machine learning," *WIREs Cognitive Science*, vol. 7, no. 2, pp. 92–111, 2016.
5. D. Nova and P. A. Estévez, "A review of learning vector quantization classifiers," *Neural Computing and Applications*, vol. 25, no. 3–4, pp. 511–524, 2014.
6. D. N. Coelho, G. A. Barreto, C. M. S. Medeiros, and J. D. A. Santos, "Performance comparison of classifiers in the detection of short circuit incipient fault in a three-phase induction motor," in *Proceedings of the 2014 IEEE Symposium on-Computational Intelligence for Engineering Solutions (CIES'04)*, 2014, pp. 42–48.
7. D. P. Sousa, G. A. Barreto, and C. M. S. Medeiros, "Efficient Selection of Data Samples for Fault Classification by the Clustering of the SOM," pp. 1–12. [Online]. Available: http://cbic2017.org/papers/cbic-paper-71.pdf
8. D. N. Coelho and C. M. S. Medeiros, "Short circuit incipient fault detection and supervision in a three-phase induction motor with a SOM-based algorithm," in *Advances in Self-Organizing Maps*. Springer, 2013, pp. 315–323.
9. D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE transactions on pattern analysis and machine intelligence*, no. 2, pp. 224–227, 1979.
10. J. C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," 1973.
11. T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.
12. P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.
13. R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. John Wiley & Sons, 2006.
14. T. Kohonen, "Improved versions of learning vector quantization," in *Proceedings of the 1990 International Joint Conference on Neural Networks (IJCNN'90)*, vol. 1, 1990, pp. 545–550.
15. A. Sato and K. Yamada, "Generalized learning vector quantization," in *Advances in neural information processing systems*, 1996, pp. 423–429.
16. B. Hammer and T. Villmann, "Generalized relevance learning vector quantization," *Neural Networks*, vol. 15, no. 8-9, pp. 1059–1068, 2002.
17. B. Hammer, M. Strickert, and T. Villmann, "On the generalization ability of GRLVQ networks," *Neural Processing Letters*, vol. 21, no. 2, pp. 109–120, 2005.