Improved Learning Rule for LVQ Based on Granular Computing

Israel Cruz-Vega(✉) and Hugo Jair Escalante

Instituto Nacional de Astrofísica, Óptica y Electrónica, 72840 Puebla, Mexico
isrcruz@ccc.inaoep.mx

Abstract. LVQ classifiers are particularly intuitive and simple to understand because they are based on the notion of class representatives (i.e., prototypes). Several approaches for improving the performance of LVQ in batch-learning scenarios are found in the literature. However, all of them assume a fixed number of prototypes in the learning process; we claim that the quantized approximation to the distribution of the input data using a finite number of prototypes, should not be fixed. Thus, in this paper we propose an improved learning algorithm for batch and on-line variants in LVQ. The proposed algorithm is based on a modified LVQ rule and granular computing, a simple and low cost computational process of clustering. All this, increases the dynamics in the learning process, proposing new prototypes which have a better covering of the distribution of classes, rather than using a fixed number of them. Similarly, in order to avoid an exponential growth in the number of prototypes, an automatic pruning step is implemented, respecting the desired reduction rate.

Keywords: LVQ · Granular computing · On-line learning

1 Introduction

Learning Vector Quantization (LVQ) is an effective technique used for supervised learning, mainly for classification purposes. The learning targets are called "prototypes", which can be understood as class representatives and yields class regions defined by hyperplanes between prototypes of the existing classes, this regions are known as Voronoi partitions [4,6].

LVQ induces efficient classifiers, i.e., they are simple and fast, as calculations are made over prototypes and not over all the neighbourhood of instances. This feature provides a great reduction in the computational cost and processing time (at least for homogeneous data sets). Usually, a fixed number of prototypes is considered during the learning process. However this is not necessarily a good way to achieve a better accuracy or faster convergence to desired theoretical values, as the Bayesian border. LVQ2.1, LVQ3 are improvements over the standard LVQ technique (also called LVQ1.0) with higher convergence speed and better approximation capabilities (related to the Bayesian borders) [7]. In general, these improvements over LVQ1.0 correct drawbacks such as: bad initialization, sensitivity to overlapping classes distribution and, instabilities due to overly pronounced elimination of dimensions in the feature space.

In this paper, based on the rule of the LVQ1.0 algorithm, we propose an improved learning algorithm for batch and on-line learning processes, where the prototypes used per each class are not fixed. Several prototypes can be produced iteratively, increasing the accuracy power of the classifier and at the same time, respecting the desired data compression rate. Also, a pruning step is proposed, based on a simple idea of a usage-frequency variable for each one of the actual prototypes; then, at each one of the iteration steps, only prototypes with more usage are kept, avoiding the excessive computational load. The proposed algorithm has some kind of memory because the usage-frequency takes into account training samples in previous samples/iterations. This is particularly important when one tries to extend the LVQ method to work on an online setting.

Two variants are proposed, "LVQ based on Granular Computing for Batch-Learning" (LVQ-GC-BL) and, "LVQ based on Granular Computing for on-line-Learning" (LVQ-GC-OL). Both methods try to overcome deficiencies of traditional LVQ-1.0 algorithm, correcting bad initialization states by means of granular computing, a simple algorithm to find centroids; and working alongside the dynamism of incremental learning, letting the algorithm not to be fixed in the number of class representatives during the learning stage. Based on this, LVQ-GC-OL is evaluated in a simulated scenario of on-line learning. It is expected that combining by the dynamic production of new instances plus the pruning step, keeping the winner prototypes per class during the online training, the algorithm will produce an acceptable trade-off between accuracy and the reduction performances, as a typical characteristic of a vector quantization algorithm.

The remainder of this paper is organized as follows: In Sect. 2, a description of the proposed algorithms for batch learning is presented. Section 3 describes the advantages of the on-line version (LVQ-GC-OL). Section 4 presents the experimental framework and reports the results obtained. Finally, Sect. 5 outlines conclusions and future work directions.

2 Improved Rule for Batch LVQ

This section introduces the proposed extension to LVQ for batch mode.

2.1 Learning Vector Quantization

Let $X = \{(\mathbf{x}_i, y_i) \subset \mathbb{R}^D \times \{1, ..., C\} \mid i = 1, ..., N\}$ be a training data set, where $\mathbf{x} = (x_1, ..., x_D) \in \mathbb{R}^D$ are D-dimensional input samples, with cardinality $|\mathbf{X}| = N; y_i \in \{1, ..., C\}\, i = 1, ..., N$ are the sample labels, and C is the number of classes. The learning structure of LVQ consists of a number of prototypes, which are characterized by vectors $\mathbf{w}_i \in \mathbb{R}^D$, for $i = 1, ..., M$, and their class labels $c(\mathbf{w}_i) \in \{1, ..., C\}$ with $Y = \{c(\mathbf{w}_j) \in \{1, ..., C\} \mid j = 1, ..., M\}$. The classification scheme is based on the best matching unit (winner-takes-all strategy), which defines the class representatives (prototypes or codebook vectors), so that the training data samples are mapped to their corresponding labels. Now, according to [8], for LVQ1: with several labeled-prototypes $\mathbf{w}_i$ of each class, the class regions in the $\mathbf{x}$ space are defined by simple nearest-neighbour comparison between $\mathbf{x}$ and the existing $\mathbf{w}_i$; the label of the closest $\mathbf{w}_i$ is the label of $\mathbf{x}$. The learning process consists in defining the optimal placement of $\mathbf{w}_i$ iteratively.

The LVQ learning tries to pull the prototypes away from the decision surfaces to demarcate the class borders more accurately. Iteratively, all of the N training instances are processed according the following rules to update the prototypes (which initially are randomly selected samples):

$$\begin{aligned}
&\mathbf{w}_c(t+1) = \mathbf{w}_c(t) + \alpha(t)\left[\mathbf{x}(t) - \mathbf{w}_c(t)\right] && \text{if } \mathbf{x}(t) \text{ is correctly classified,}\\
&\mathbf{w}_c(t+1) = \mathbf{w}_c(t) - \alpha(t)\left[\mathbf{x}(t) - \mathbf{w}_c(t)\right] && \text{if the classification of } \mathbf{x}(t) \text{ is incorrect,}\\
&\mathbf{w}_i(t+1) = \mathbf{w}_i(t) \ \text{ for } i \neq c && \text{for prototypes of other classes}
\end{aligned} \tag{1}$$

where $\mathbf{w}_c(t)$ is the closest prototype to $\mathbf{x}(t)$ at iteration t in the Euclidean metric, $\alpha(t)$ is a scalar gain $(0 < \alpha < 1)$, decreasing monotonically in time.

2.2 Granular Computing

In this paper, granules play the role of prototypes. The idea of granular computing, has been developed in previous works [1,2]. In these works, the granular computing generates surrogate models in expensive fitness functions of optimization problems; then, not only the core of information is drastically reduced, but also, a basic knowledge of the structure of the model is obtained [3]. In the following we detail the granule generation process. The algorithm starts by selecting random a training sample as the center of the granule for each one of the classes. Each one of these granule-prototypes are represented by density functions. Next, for each one of the elements of the data set $(\mathbf{x}_i, y_i)$, a measure of "similarity" to the existing prototypes is computed using the following Gaussian similarity measure:

$$\mu_k(\mathbf{x}_j) = \exp\left(\frac{-\left(\mathbf{x}_j^i - \mathbf{w}_k^i\right)^2}{\sigma_k^2}\right) \tag{2}$$

where $\mathbf{w}_k$ is the center of the k^{th} granule or the prototype, for $k = 1, 2, ..., l$ number of granules, and $\mathbf{x}_j$ is the j^{th} input of the training set, that belongs to the i^{th} class. The radius σ_k of each granule is used to control the area of similarity degree between inputs, determining the decision boundary of inclusion into the granules. In this work, this radious σ_k is formulated as the mean value of the Euclidean distance between a granule and all the inputs,

$$\sigma_{k_i} = \frac{1}{n}\sum_{j=1}^{n}\sqrt{\left(\mathbf{x}_j - \mathbf{w}_k\right)^2} \tag{3}$$

where n is the number of inputs of the training set.

Now, let

$$c = \arg\max\left\{\mu_k(\mathbf{x}_j)\right\} \tag{4}$$

which defines the nearest $\mathbf{w}_i$ granule to $\mathbf{x}$, denoted by $\mathbf{w}_c$. Initially we have a granule for each one of the classes of the training set.

2.3 Improved Batch-Learning Rule

The proposed algorithm is designed to create new granules or prototypes and update the existing ones according to the measure of similarity mentioned above and a threshold θ. The following equations define our improved LVQ-1.0 process:

$$\begin{aligned} R_1 : &\mathbf{w}_c\left(t+1\right) = \mathbf{w}_c\left(t\right) + \alpha\left[\mathbf{x}_j^i\left(t\right) - \mathbf{w}_c\left(t\right)\right] \\ &\text{if } \mathbf{x}_j^i \text{ and } \mathbf{w}_c \text{ belongto the same class and } c \geq \theta, \end{aligned} \tag{5}$$

$$\begin{aligned} R_2 : &\mathbf{w}_c\left(t+1\right) = \mathbf{w}_c\left(t\right) - \alpha\left[\mathbf{x}_j^i\left(t\right) - \mathbf{w}_c\left(t\right)\right] \\ &\text{if } \mathbf{x}_j^i \text{ and } \mathbf{w}_c \text{ belong to different classes} \end{aligned} \tag{6}$$

$$\begin{aligned} R_3 : &\text{Add a new } \mathbf{w}_i \text{ of the same class of } \mathbf{x}_j^i \\ &\text{if } \mathbf{x}_j^i \text{ and } \mathbf{w}_c \text{ belong to the same class, but } c < \theta \end{aligned} \tag{7}$$

where $\mathbf{w}_c$ represents the nearest prototype to the actual input sample $\mathbf{x}_j^i$, α represents the learning rate, and $0 < \alpha < 1$, also α will decrease monotonically with time by each iterative step. When each one of the sample input $\mathbf{x}_j^i$ of the data set arrives, it is measured by (2) to each one of the $\mathbf{w}_i$ elements, then, it is obtained the closer one, $\mathbf{w}_c$. The algorithm will perform the conventional LVQ-1.0 rule (R_1, R_2), if the measure of similarity exceeds θ, that is, the sample input $\mathbf{x}_j^i$ is not only near to any one of the prototypes and has the same class, but also, is within the receptive field of the granule. If $\mathbf{x}$ is out of this receptive field, that is, $\mu_k\left(\mathbf{x}_j^i\right) < \theta$, a new prototype will be added to the existing ones. In order to avoid increasing the number of number of prototypes without control, a pruning step will be developed in each iteration step. A better explanation of this pruning step is given in the next subsection.

Now, considering the threshold value θ, by which new prototypes are added to the pool of the existing ones W_k, we proceed as follows. First, note that all the process is performed in the hypercube $[0,1]$. Then, since the measure of closeness between the input values $\mathbf{x}_j^i$ and the existing prototypes is given by a Gaussian measure of similarity (2), the nearest prototype $\mathbf{w}_c$ has a value close to 1 and, as the prototype is further away, the value will be near of 0. We want a dynamically process which gives the algorithm the possibility to create new prototypes every iteration, then, this value is near 1; that is, the receptive field of $\mathbf{w}_c$ is too small in the hypercube $[0,1]$.

2.4 Pruning Step

Avoiding an exponential growth in the number of prototypes, each one of the prototypes of all classes will compete for their survival through a life index L_k. The pruning step of prototypes will be done in each iteration step. The prototypes that have the highest L_k-value for each one of the existing classes C will survive, and the others will be eliminated, according to a desired reduction rate. Hence, L_k is initially set at 1 for each one of the initial prototypes and subsequently updated as below:

$$L_k^i = \begin{cases} L_k^i + 1 & \text{If } k = K \\ L_k^i & \text{Otherwise} \end{cases} \tag{8}$$

where the constant 1 is the life reward for the winning prototype $\mathbf{w}_c$ and K is the index of this selected prototype, according to (2) and (4).

With this, the life index L_k attached to each granule that was not pruned, past information is maintained for the next iterative steps from the last ones. The flow chart of the LVQ-GC-BL algorithm is shown in Fig. 1.

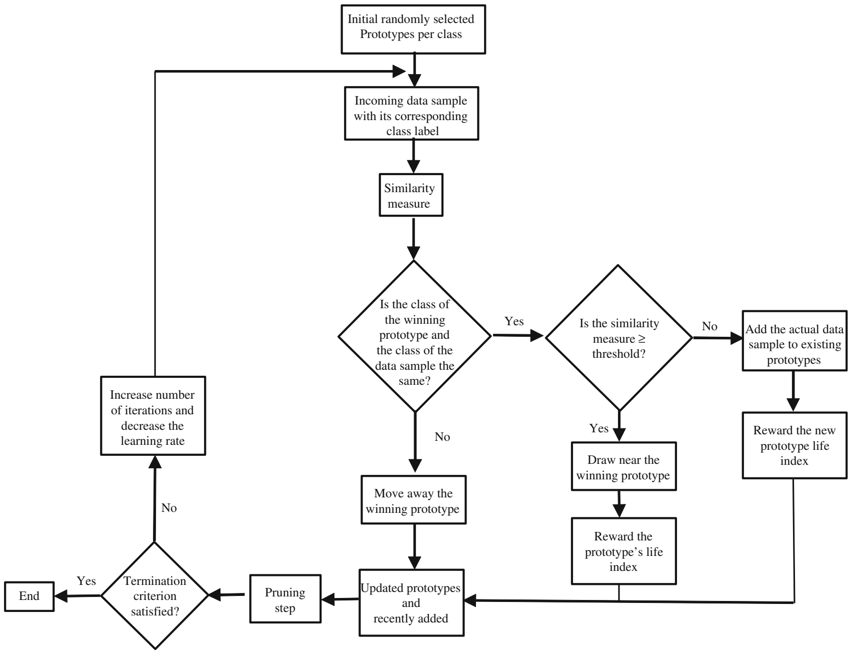
Initial randomly selected Prototypes per class
Incoming data sample with its corresponding class label
Similarity measure
Is the class of the winning prototype and the class of the data sample the same?
Yes
Is the similarity measure ≥ threshold?
No
Add the actual data sample to existing prototypes
Reward the new prototype life index
Yes
Draw near the winning prototype
Reward the prototype's life index
No
Move away the winning prototype
Updated prototypes and recently added
Pruning step
Termination criterion satisfied?
Yes
End
No
Increase number of iterations and decrease the learning rate

Fig. 1. LVQ-GC-BL algorithm.

3 On-line Algorithm

The online version (LVQ-GC-OL), generates a model from the training set, seeing each sample point once. That is the LVQ process the N training samples a single time. The improved LVQ not only updates granules but also allows the creation of new ones, also it is equipped with memory and punning mechanisms. Therefore, we can say that the advantages of the on-line version of our proposal are:

1. The learning process is done on-line, during every incoming sample, and the algorithm does not require of the storage of certain number of instances to perform the learning process.

2. Each prototype used and newly created, due to the attached life index to it, contains a kind of "*memory*" or "*weight*" associated to each prototype during the learning process. Once the learning process has finished (in a single iteration), the more used class representatives prototypes will be kept.
3. At every incoming sample, due to the receptive field of the prototype and the proposed threshold, not only the actual prototypes will be updated, but also the algorithm can "propose" new ones. With this, the quantized approximation to the real density distribution of classes has more "proposed" elements to cover it.

4 Experimental Framework

The experimental part of the proposed method is developed using the suite of data sets introduced in [18]. This benchmark consists of 59 data sets associated to different classification problems. The results are reported separately for small (data sets with less than $2,000$ instances) and large (data sets with at least $2,000$ instances) data sets. The evaluation methodology adopted consists of applying Prototype Generation (PG) methods to each data set in a 10-fold cross validation scheme. For the evaluation process, a 1NN rule is used by the generated prototypes. The following experimental settings, also used in [18], are defined to assess the proposed approach: 1 initial prototype per class; $Iterations = 100$; $\theta = 0.9; \alpha = 0.1$.

Now we report experimental results to show the effectiveness of the proposed method LVQ-GC-BL. The performance of LVQ-GC-BL is compared against other techniques for PG mentioned in [18], evaluated with the same benchmark conditions.

4.1 Classification Performance of Prototypes

Table 1 shows the average classification performance obtained by the LVQ-GC-BL algorithm for small and large data sets, and is compared against LVQ3 and GENN, the last one is the method that obtained the highest accuracy in [18].

Comparisons from Table 1, shows the effectiveness of our LVQ-GC-BL method in terms of classification performance. We have to remember that LVQ3 algorithm is an improvement over LVQ1.0 algorithm (by which we are basing our proposal). In this work we are improving the LVQ1.0 algorithm by making it variable in the number of class representatives. Which results in improved performance of LVQ.

Table 1. Average classification accuracy for LVQ-GC-BL and reference methods

	LVQ-GC-BL	LVQ3	GENN
Small	0.7038 ± 0.0755	0.6930 ± 0.1560	0.756 ± 0.05
Large	0.7517 ± 0.2068	0.7318 ± 0.2093	0.813 ± 0.217

GENN obtained better values in classification performance, but its reduction performance is among the worst (see below). One should note that not only a high classification performance is desired, also the data compression rate is of main concern. Next, we show results in this important parameter.

4.2 Reduction Performance of Prototypes

Table 2 shows the average instance-reduction rates comparing our algorithm LVQ-GC-BL against LVQ3 and GENN algorithms. In this paper, we consider a percentage of reduction with respect to the data set size of at least 95 %, this is according to similar reduction rates for the best algorithms in [18].

Table 2. Average reduction rates for LVQ-GC-BL and reference methods

	LVQ-GC-BL	LVQ3	GENN
Small	0.9542 ± 0.0154	0.9488 ± 0.0083	0.1862 ± 0.1206
Large	0.9688 ± 0.0180	0.9799 ± 0.0008	0.1576 ± 0.1992

We can see that our algorithm LVQ-GC-BL is highly competitive in compression of the data set. For small data sets, presents the better results and here, we can verify that, although GENN is better in classification performance, is the worst for data compression rate.

4.3 Execution Time

Next, the execution time of simulation comparing LVQ3, GENN and our algorithm is shown in Table 3. This Table shows that our algorithm has the better execution time for large data sets, and is slightly improved in execution speed by the LVQ-3 algorithm for small data sets. Our algorithm, due to the pruning step at each iteration, avoids excessive computational load in the training process.

Table 3. Execution time for LVQ-GC-BL and reference methods

	LVQ-GC-BL	LVQ3	GENN
Small	0.373	0.2316	1.4285
Large	0.541	1.7037	167.4849

4.4 On-line Learning

For the online version of the algorithm (LVQ-GC-OL), the learning process will be developed in just one iteration step (i.e., $Iterations = 1$), the learning rate is fixed and does not decrease, the rest of the parameters remained the same as above. Results of the classification performance are shown in Table 4.

Table 4. Average classification accuracy and execution time for LVQ-GC-OL

	Avg. classification accuracy	Avg. execution time
Small	0.67257 ± 0.16036	0.05017
Large	0.72264 ± 0.19731	0.06857

Although, in general the average of classification accuracy is better for batch learning rather than the online approach, in many cases, and for particular data sets, the accuracy was better in the on-line learning. This improvement in the accuracy for such particular data sets could be done due to the dynamism of the proposed algorithm. That is, every time an instance is evaluated against the existing prototypes, according to the proposed algorithm, not only the learning process can update these prototypes, but also can create new ones if the receptive field does not cover the distribution area of the corresponding class. Then, increasing the number of prototypes provides a better coverage in the class-distribution area, rather than just move a non incremental and limited number of initial prototypes according to the traditional rules. At the same time, the pruning step prevents a non-controlled increase of these class representatives.

4.5 Visualizing Learned Prototypes for a 2-D Case

The visualization of learned prototypes gives a better understanding of prototypes distribution in the training set space. For a 2-Dimensional case, the banana data set provides this possibility for visualization purposes. This data set contains 5300 instances described by 2 attributes, and for classification purposes only has 2 classes. Figure 2 shows the results of the learned prototypes for our LVQ-GC-BL (left) algorithm and the comparative LVQ-3 method (right). For comparison purposes, the training parameters are: 20 initial prototypes per class; $Iterations = 100$; $\theta = 0.7; \alpha = 0.1$.

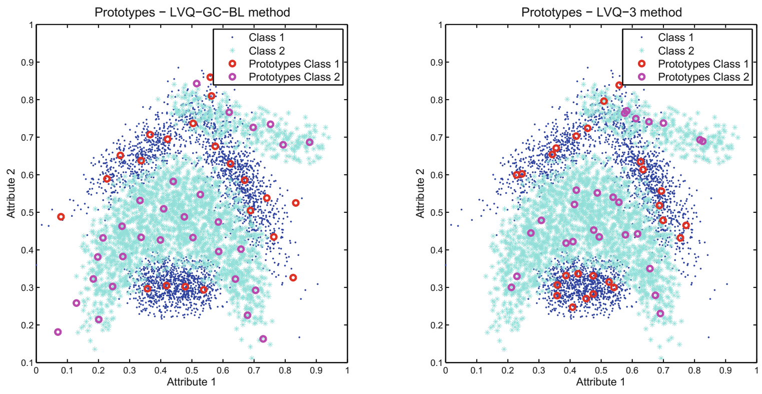

Fig. 2. Banana data set and prototypes

From Fig. 2 it can be seen that Prototypes produced by the LVQ-GC-BL algorithm, have better classes distribution than those of LVQ3, prototypes in LVQ-GC-BL, are not to close between them (i.e., they are spread through the input space), and prototypes generated by LVQ-GC-BL are covering areas that the LVQ 3 algorithm does not. These advantages are due to the dynamism in the production of new prototypes and the pruning step at every iteration. According to the proposed threshold and the life index attached to each prototype, the algorithm will only update prototypes which are more useful and will produce new ones in order to cover different zones than the existing ones by the actual prototypes. With this, a better approximation to the correct class distribution of classes is achieved.

5 Conclusion

This work presents an improved version of the classical LVQ-1.0 algorithm. The algorithm is compared against relevant classifiers such as LVQ-3 and GENN, showing acceptable values of accuracy and data compression rate. The main contributions of the algorithm can be stated as follows: (1) based on granular computing, we have a better initialization of the prototypes; which avoids the use of other algorithms like k-means clustering or SOM, (2) The proposed improvement is able to generate new prototypes at every iterative step, then, not only the actual prototypes are updated but also the new ones can have a better covering of the density distribution class, (3) the proposed life index, attached to every prototype, helps to keep the better class representatives during each iterative step and, this is a measure of "memory" persisting during the learning cycle. With all these steps, the algorithm is able to find the best location for the prototypes and, provides of a better approximation to the correct density distribution classes.

For the on-line version of the algorithm, due to the dynamism in the production of new prototypes and the update of the existing ones at every sample point, we can see that the algorithm produces an acceptable level of accuracy. The speed of the algorithm and the pruning step, makes it ideal to work in environments of on-line data sets, where information is constantly flowing.

Future work is mainly oriented to propose an adaptive receptive field and threshold value, which will generate prototypes according to the specific data features presented.

References

1. Akbarzadeh-T, M.R., Davarynejad, M., Pariz, N.: Adaptive fuzzy fitness granulation for evolutionary optimization. Int. J. Approximate Reasoning **49**(3), 523–538 (2008)
2. Cruz-Vega, I., Garcia-Limon, M., Escalante, H.J.: Adaptive-surrogate based on a neuro-fuzzy network and granular computing. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, pp. 761–768. ACM (2014)

3. Cruz-Vega, I., Escalante, H.J., Reyes, C.A., Gonzalez, J.A., Rosales, A.: Surrogate modeling based on an adaptive network and granular computing. Soft Comput. 1–15 (2015)
4. Nova D., Estévez, P.A.: A review of learning vector quantization classifiers. Neural Comput. Appl. 1–14 (2013)
5. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 139–148. ACM (2009)
6. Kohonen, T., Maps, S.-O.: Self-organizing Maps. Springer Series in Information Sciences, vol. 30. Springer, Heidelberg (1995)
7. Kohonen, T.: Improved versions of learning vector quantization. In: 1990 International Joint Conference on Neural Networks, 1990 IJCNN, pp. 545–550. IEEE (1990)
8. Kohonen, T.: The self-organizing map. Proc. IEEE **78**(9), 1464–1480 (1990)
9. Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Friedman, J., Tibshirani, R.: The Elements of Statistical Learning, vol. 2, 1st edn. Springer, New York (2009)
10. Garcia, S., Derrac, J., Cano, J.R., Herrera, F.: Prototype selection for nearest neighbor classification: taxonomy and empirical study. IEEE Trans. Pattern Anal. Mach. Intell. **34**(3), 417–435 (2012)
11. García-Limón, M., Escalante, H.J., Morales, E., Morales-Reyes, A: Simultaneous generation of prototypes and features through genetic programming. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation, pp. 517–524. ACM (2014)
12. Bharitkar, S., Filev, D.: An online learning vector quantization algorithm. In: ISSPA, pp. 394–397 (2001)
13. Wang, J.-H., Sun, W.-D.: Online learning vector quantization: a harmonic competition approach based on conservation network. IEEE Trans. Syst. Man Cybern. Part B Cybern. **29**(5), 642–653 (1999)
14. Poirier, F., Ferrieux, A.: {DVQ}: dynamic vector quantization-an incremental {LVQ} (1991)
15. Lughofer, E.: Evolving vector quantization for classification of on-line data streams. In: 2008 International Conference on Computational Intelligence for Modelling Control & Automation, pp. 779–784. IEEE (2008)
16. Zang, W., Zhang, P., Zhou, C., Guo, L.: Comparative study between incremental and ensemble learning on data streams: case study. J. Big Data **1**(1), 5 (2014)
17. Zadeh, L.A.: Some reflections on soft computing, granular computing and their roles in the conception, design and utilization of information/intelligent systems. Soft Comput. Fusion Found. Methodol. Appl. **2**(1), 23–25 (1998)
18. Triguero, I., Derrac, J., Garcia, S., Herrera, F.: A taxonomy and experimental study on prototype generation for nearest neighbor classification. IEEE Trans. Syst. Man Cybern. Part C Appl. Rev. **42**(1), 86–100 (2012)