Applied Intelligence (2020) 50:1590–1608
https://doi.org/10.1007/s10489-019-01591-5

Neuroevolutionary learning in nonstationary environments

Tatiana Escovedo[1] · Adriano Koshiyama[2] · Andre Abs da Cruz[3] · Marley Vellasco[1]

Published online: 30 January 2020

Abstract
This work presents a new neuro-evolutionary model, called NEVE (Neuroevolutionary Ensemble), based on an ensemble of Multi-Layer Perceptron (MLP) neural networks for learning in nonstationary environments. NEVE makes use of quantum-inspired evolutionary models to automatically configure the ensemble members and combine their output. The quantum-inspired evolutionary models identify the most appropriate topology for each MLP network, select the most relevant input variables, determine the neural network weights and calculate the voting weight of each ensemble member. Four different approaches of NEVE are developed, varying the mechanism for detecting and treating concepts drifts, including proactive drift detection approaches. The proposed models were evaluated in real and artificial datasets, comparing the results obtained with other consolidated models in the literature. The results show that the accuracy of NEVE is higher in most cases and the best configurations are obtained using some mechanism for drift detection. These results reinforce that the neuroevolutionary ensemble approach is a robust choice for situations in which the datasets are subject to sudden changes in behaviour.

Keywords Concept drift · Adaptive learning · Nonstationary environments · Neuroevolutionary ensemble · Quantum-inspired evolution

1 Introduction

The ability of a classifier to learn from incremental and dynamic data extracted from a nonstationary environment (when data distribution changes over time) poses a challenge to the field of computational intelligence. In the context of neural networks, the problem becomes even more complicated, since most of the existing models must be retrained when a new data block is available, using the whole set of patterns learned until then. To cope with that sort of problem, a classifier must, ideally, be able to [43]:

- Track and detect any changes in the underlying data distribution;
- Learn with new data without the need to present the whole dataset again for the classifier;
- Adjust its own parameters in order to address the detected changes on data;
- Forget what has been learned when that knowledge is no longer useful for classifying new instances.

All these abilities seek, in one way or another, to deal with a phenomenon called concept drift [51, 22]. This phenomenon defines datasets that suffer changes over time, such as when there is a change in the relevance of the variables, or when the mean and variance of the variables change.

Many approaches have been devised to accomplish some or all of the abilities mentioned above. One of the older and simpler approaches is a sliding window (not always continuous) on the input data used to train the classifier with the data delimited by this window [21]. Another method is to detect deviations and, if they occur, to adjust the classifier [7]. Some models, in turn, use rule-based classifiers, like [43, 59–61]. A more successful and widely used approach though is to use a group of different

✉ Adriano Koshiyama
a.koshiyama@cs.ucl.ac.uk

Tatiana Escovedo
tatiana@inf.puc-rio.br

Andre Abs da Cruz
andrevpuc@gmail.com

Marley Vellasco
marley@ele.puc-rio.br

[1] Department of Electrical Engineering, PUC-Rio, R. Marquês de São Vicente, 225 - Gávea, Rio de Janeiro 22430-060, Brazil

[2] Department of Computer Science, University College London, London, UK

[3] MDC Partners, Antwerp, Belgium

classifiers (ensemble) to cope with changes in the environment. Several different ensemble models have been proposed in the literature, including recent approaches like [56–58], and may or may not weigh each of its members. Most models using weighted classifier ensembles determine the weights for each classifier using a set of heuristics related to classifier performance in the most recent data received [22].

Although several algorithms have already been proposed in the literature for classification in concept drift scenarios - many even using ensembles - for this type of problem, neuroevolution has still been little explored. Neuroevolution uses evolutionary algorithms to adjust parameters that affect the performance of artificial neural networks, such as topology, learning rate, weights, among others. In this case, each solution of the evolutionary algorithm stores a representation of these parameters, which are evolved to find the optimal network for the problem. Applied to neural network ensembles, evolutionary algorithm is also able to dynamically adjust the entire model, a task that would be very arduous if performed manually, due to the complexity of the model.

Because of the architecture complexity, it is necessary that the neuroevolutionary models based on classifier ensembles have good computational performance and fast convergence, in order to be able to be applied in real scenarios. This feature becomes even more relevant in nonstationary environments, since it is necessary to update the ensemble each time new data become available or when some change is detected in data. Thus, this step must be fast so as not to compromise the overall performance of the model. To deal with this issue, an interesting and still little-explored strategy in the literature related to neuroevolutionary models is the quantum-inspired evolutionary algorithms. This is a class of evolutionary algorithms developed to achieve better performance in computationally intensive problems, inspired by quantum computing principles [17, 18, 2, 39, 52, 8]. One of the main advantages of the quantum-inspired evolutionary models is that good solutions are obtained with the smallest possible number of evaluations. This class of algorithms has been previously used in the literature to solve combinatorial and numerical optimization problems, based on binary [18, 39] and real representations [2, 39, 52], providing better results and using less computational effort than classical genetic algorithms [47]. Applied to neural network ensembles, quantum-inspired evolutionary algorithms can be used to model the neural networks and to determine the voting weights for each ensemble member. Thus, each time a new block of data arrives, the ensemble can be optimized, improving its classification performance for the new data.

Models for learning in nonstationary environments can or cannot contain drift detection mechanisms. Most of the models found in the literature assume that the changes occur in a hidden context external to the model itself and, therefore, the drift cannot be predicted [15]. For this reason, these models use the passive and reactive approaches, that is, from the results of the model (in classification problems, the label predicted by the model is compared with the real label received), verify the drift occurrence and react to it only after the error is observed in the model. However, anticipating the detection of drift in the input data before they are submitted for prediction (i.e., before receiving the true labels) seems to be a more satisfactory approach since it permits to adjust the model previously to better deal with the new scenario and avoid the classification error. For this reason, the model proposed in this work uses this active approach, being an important differential compared to the existing approaches in the literature.

Given the above, the main objective of this work is to propose and develop a self-adaptive and flexible model, with good accuracy and suitable for learning in nonstationary environments. A new quantum-inspired neuroevolutionary model, based on a Multi-Layer Perceptron (MLP) neural network ensemble, will be presented for learning in nonstationary environments. The proposed model, called NEVE (Neuroevolutionary Ensemble), has the following characteristics:

- Contains a concept drift detection mechanism, with the ability to detect changes proactively or reactively. This method, already detailed in [10] allows the reaction and adjustment of the model whenever necessary;
- Performs the automatic generation of new classifiers for the ensemble, most suitable for the new input data, using the quantum-inspired evolutionary algorithm for numerical and binary optimization (QIEA-BR) [39];
- Automatically determines the voting weights of each ensemble member, using the quantum-inspired evolutionary algorithm for numerical optimization (QIEA-R) [2, 52], a simplified version of QIEA-BR.

Several experiments were performed with artificial and real datasets to validate and compare the performance of the proposed model with other existing models for learning in nonstationary environments, verifying how the detection model affects the performance and accuracy of NEVE.

This work is structured in four additional sections. Section 2 presents a brief review of the literature related to the fundamentals of concept drift. It also describes the evolutionary models with quantum inspiration used in this work: QIEA-R and QIEA-BR. Section 3 presents the proposed neuroevolutionary model (NEVE) and Section 4 discusses the experimental results. Finally, Section 5 presents the conclusions of this work and possibilities of future work.

2 Literature review

2.1 Concept drift

The term concept drift can be defined informally as a change in the concept definition over time and, hence, change in its

distribution. Concept drift refers to a supervised learning scenario, where the relationship between the input data and the target variable changes over time [15]. An environment from which this kind of data is obtained is considered a nonstationary environment. Formally speaking, considering the posterior probability of a sample x belonging to a class y, according to [9] concept drift is any scenario in which this probability changes over time, that is: $P_{t+1}(y|x) \neq P_t(y|x)$.A practical example of concept drift mentioned in [29] is detecting and filtering out spam e-mails. The description of the two classes "spam" and "non-spam" may vary over time. They are user specific, and user preferences also change over time. Moreover, the variables used at time t to classify spam may be irrelevant at t + k. In this way, the classifier must deal with "spammers", who will keep creating new forms to trick the classifier into labeling a spam as a legitimate e-mail.

Concept drift is usually classified in abrupt or gradual [15, 51, 54]. The abrupt drift occurs when a concept A is abruptly switched for another concept B, that is, at time t the source S1 is suddenly replaced by S2. The gradual drift, on the other hand, happens when a concept A is gradually exchanged for the other concept B. In this case, while there is no definitive change from concept A to concept B, we observe more and more occurrences of B and fewer occurrences of A. Both sources S1 and S2 are active, but as time passes, the probability of sampling the source S1 decreases as the sample probability of the source S2 increases. At the beginning of this drift, before more instances are observed, an instance of the S2 source can be easily mistaken for random noise. It is important to note that noise (or outlier) is not considered a type of drift because it refers to an anomaly or isolated occurrence of a random drift. In this case, there is no need to adapt the model, which should be robust to noise.

The term "Drift Detection" refers to techniques and mechanisms for detecting drift by identifying points of change or small intervals during which the variations occur. In this case, the environment has sufficiently changed so that the existing models can no longer be effective to predict the behavior of the current data [15]. Several drift detection mechanisms have already been proposed in the literature, but most of them work reactively: they compare the class predicted by the classifier to the correct class label received later, noticing the drift only after its occurrence and the misclassification. Only then, the reactive detector applies a sequence of procedures to identify some change in the conditional class distribution - a concept drift. Examples of reactive detectors can be found in [14, 5, 36, 4, 42, 3, 31, 23, 13, 46].

Few papers use a proactive approach. [28] applies principal component analysis (PCA) for features extraction before the drift detection. The authors discuss and show evidence that components with lower variance should be stored as the extracted features, since they are more likely to be affected by a change. The authors then choose a change detection criterion based on the semiparametric log-likelihood function that is sensitive to changes in the mean and variance of the multidimensional distributions.

In [10], we proposed a new drift detection mechanism, called DetectA (Detect Abrupt Drift), which uses a proactive detection approach. This model is used in the experiments of this work and comprises three basic steps: (i) label the patterns from the test set (an unlabelled data block), using an unsupervised method; (ii) compute some statistics from the training and test sets, conditioned to the given class labels provided in the training set; and (iii) compare the training and testing statistics using a multivariate hypothesis test. Based on the results of the hypothesis tests, we attempt to detect the drift on the test set, before the real labels are obtained.

Algorithms for handling concept drift problems can be categorized in several ways. Table 1, based on [9, 27, 29, 30], summarizes the most commonly used classifications in the literature, with their respective definitions.

Algorithms that use the passive approach (without drift detection) regularly update the model as new data arrives and a forgetting heuristic is used, independently of the existence of change. For example: in a classifier ensemble, the weights of the members are updated after each new data received (individual or in blocks), based on the recent accuracy of ensemble members. Without concept drift, the classification accuracy will be stable and the weights will converge. If any changes occur, the weights will change to reflect them, without the need for explicit detection [29].

However, this can be very costly if the amount of data that arrives is excessively large or if the application require user feedback to label the data, which can be time-consuming. One way to reduce this problem is to use special techniques to detect changes and adapt the model only when unavoidable, using the active approach [51], also called trigger approach. In general, when active approaches detect a drift, some action is taken, for example, configuring a window with the latest data and retraining the classifier, or adding a new classifier to the ensemble.

Thus, the active method seeks to point out when the drift occurred and allows the model to modify itself or continue learning in the same way. A disadvantage of this method is the risk of having an imperfect mechanism that can produce false alarms, which is very common particularly in noisy datasets. In the passive mechanism, the learner believes that the environment can change at any time or can be continuously changing. The algorithm then continues to learn from the environment, building and organizing its knowledge base. If a change has occurred, it is learned. If nothing happened, the existing knowledge is reinforced [9]. The majority of literature ensembles follow a passive schema of adaptation, whereas active approaches are usually used with single online classifiers [27]. The models [24, 26, 48] are examples of passive approaches and the models [14, 5, 36–38, 32] are examples of active approaches.

Table 1 – Types of Algorithms

Passive x Active Approach	
Passive	Assume possibly ongoing drift and continuously update the model with each new data (set). If a change has occurred, it is learned; else, the existing knowledge is reinforced.
Active	Uses some drift detection mechanism, learning only when the drift is detected.
Individual input x Input in Blocks	
Individual	Learn one instance at a time. They have better plasticity but poorer stability properties. They also tend to be more sensitive to noise as well as to the order in which the data are presented.
In blocks	Requires blocks of instances to learn. They benefit from the availability of larger amounts of data, have better stability properties, but can be ineffective if the batch size is too small, or if data from multiple environments are present in the same batch. Typically use some form of windowing to control the batch size.
Single Classifier x Ensemble	
Single classifier	Uses only one classifier.
Ensemble	Combines multiple classifiers.

Regarding data entry, it is worth emphasizing that individual patterns can be converted into batches or blocks of data. The opposite is also possible, but block data can come in large quantities, making instance-based processing very time-consuming [29].

Comparing single classifier x ensemble approaches, ensemble-based approaches are newer and tend to have better accuracy, flexibility, and efficiency than those using a single classifier [29]. It is important to remember that in massive datasets it is often preferred to use simple models - such as single classifiers - since there may not be time to execute and update an ensemble. On the other hand, some authors argue that a simple ensemble may be easier to use than certain simple adaptive classifiers, such as decision trees. When time is not the main concern, but high accuracy is required, an ensemble becomes the natural solution. For example, in mammography screening for tumors, it is acceptable to take a few minutes per image [30]. Ensemble approaches can use different methods to adapt to a concept drift.

As mentioned earlier, responding to several types of concept drift is a difficult task for a simple classifier. For this reason, several systems based on classifier ensembles have recently been proposed to deal with concept drift learning, such as [49, 48, 11, 12, 44, 24–26, 45, 9, 33, 53, 6, 50]. The main novelty proposed in this work is the possibility of using an active drift detection mechanism (DetectA) together with an ensemble of neural networks, trained and combined through quantum-inspired evolutionary algorithms, allowing automatic and dynamic adjustment of the classifiers and their weights in the ensemble, using less computational time.

2.2 Quantum-inspired evolutionary algorithms

Classical evolutionary algorithms have been used successfully to solve complex optimization problems in a wide range of fields, such as automatic circuit design and equipment, task planning, software engineering and data mining, among many others [1, 2]. The fact that this class of algorithms does not require rigorous mathematical formulations about the problem to be optimized, besides offering a high degree of parallelism in the search process, are some of the advantages of the use of evolutionary algorithms.

However, some problems are computationally costly regarding the evaluation of the fitness function during the search process, making optimization by evolutionary algorithms a slow process for situations where a fast response is desired (as in online optimization problems). In order to address these issues, quantum-inspired evolutionary algorithms have been developed, which are a class of estimation distribution algorithms that perform better in combinatorial and numerical optimization when compared to their homologous canonical genetic algorithms [1, 2, 8, 17, 18, 39, 52].

These algorithms are inspired by concepts of quantum physics, in particular in the concept of superposition of states, and were initially developed for optimization problems using binary representation, such as the Quantum-Inspired Evolutionary Algorithm (QIEA-B) [17–20], which uses a chromosome formed by q-bits. Each q-bit consists of a pair of numbers (α, β), where $|\alpha^2| + |\beta^2| = 1$. The value $|\alpha^2|$ indicates the probability that the q-bit has value 0 when observed, while value $|\beta^2|$ indicates the probability that the q-bit has value 1 when observed. Thus, in QIEA-B, a quantum individual is formed by M q-bits, according to (1):

$$\left|\begin{matrix}\alpha_{i1}\\ \beta_{i1}\end{matrix}\right\| \left.\begin{matrix}\alpha_{i2}\\ \beta_{i2}\end{matrix}\right| \ldots \left|\begin{matrix}\alpha_{iM}\\ \beta_{iM}\end{matrix}\right| \tag{1}$$

where i = 1, 2, 3, ..., M.

Quantum-inspired evolutionary algorithms were then extended to real representation, to better deal with numerical optimization problems. In these problems, the direct representation is more appropriate, in which real numbers are directly encoded in a chromosome rather than converting binary strings into numbers. With real numerical representation, the memory demand is reduced while the precision is increased [1]. Thus, the Quantum-Inspired Evolutionary Algorithm with Real Representation (QIEA-R) was developed [1, 2], inspired by the concept of multiple universes of quantum physics. In this scenario, the algorithm allows performing the optimization process with a smaller number of evaluations, substantially reducing the computational cost. Next sections describe the QIEA-R and QIEA-BR models, which are better suited to neuroevolution.

2.2.1 Quantum-inspired evolutionary algorithm with real representation (QIEA-R)

Originally proposed in [1], this algorithm was used to solve numerical optimization benchmark problems and the neural evolution of recurrent neural networks. The results obtained demonstrated the efficiency of this algorithm in the solution of these types of problems.

In QIEA-R, the quantum population Q(t) consists of N quantum individuals qi (i = 1, 2, 3, .., N) which are composed of G quantum genes. Each quantum gene is formed by a probability density function (PDF), which represents the superposition of states and is used to observe the classical gene. Quantum individuals can be represented by:

$$q_i = [g_{i1} = p_{i1}(x), g_{i2} = p_{i2}(x), ..., g_{iG} = p_{iG}(x)] \quad (2)$$

where i = 1, 2, 3, ..., N, j = 1, 2, 3, ...,G and pij functions represent the probability density functions used by the QIEA-R to generate the values for the genes of the classical individuals. In other words, the pij(x) function represents the probability density of observing a given value for the quantum gene when its overlap is collapsed. The probability density function used by [1] is the square pulse, an uniform function of simple geometry, which can be defined by eq. 3:

$$p_{ij}(x) = \begin{cases} U_{ij} - L_{ij}, L_{ij} \leq x \leq U_{ij} \\ 0, otherwise \end{cases} \quad (3)$$

where Lij is the lower limit and Uij is the upper limit of the interval in which the gene j of the i-th quantum individual can collapse, i.e., assume values when observed.

For the case where pij(x) is a square pulse, the quantum gene can be represented by storing the position of the center point of the square pulse and its width: μ_{ij} and σ_{ij}, respectively. The QIEA-R also uses a population of quantum individuals, which are observed to generate the classical individuals. The updating of the quantum individuals is carried out based on the evaluation of the classic individuals: μ_{ij} and σ_{ij} are altered in order to bring the pulse to the most promising region of the search space, increasing the probability of observing a certain set of values for the classical gene in the vicinity of the most successful individuals in the classical population. The pseudocode of the QIEA-R algorithm is shown in Appendix 1.

In this work, the QIEA-R is used to evolve voting weights for each classifier member of the ensemble and thus determine the final decision of the ensemble. In this way, the chromosome will have size n, where n represents the number of ensemble members. Each gene, in turn, will represent the voting weight associated with each classifier. Further details on QIEA-R can be found in [1, 2, 52].

2.3 Quantum-inspired evolutionary algorithm with binary-real representation (QIEA-BR)

The main motivation for creating an algorithm with mixed representation is that many real problems cannot be solved only by numerical decisions or combinatorial decisions. More specifically in the field of neural networks, the modeling process may involve combinatorial decisions (selection of the most relevant variables to the input layer, how many neurons should be used in the middle layer, etc.) and, simultaneously, numerical decisions (optimal values for synaptic weights).

With this motivation, [40] proposed the creation of an algorithm with quantum inspiration and binary-real representation, called QIEA-BR, for simultaneous optimization of combinatorial and numerical problems, that is, of mixed nature. The QIEA-BR algorithm was the first evolutionary algorithm with quantum inspiration and mixed representation proposed in the literature and will inherit the main characteristics of its precursors, such as global problem-solving ability and probabilistic representation of the search space. This mixed representation results in high population diversity in each quantum individual and the need of fewer individuals in the population to explore the search space.

The QIEA-BR algorithm also requires a population of quantum individuals that represents the overlap of possible states that the classical individuals can assume when observed. The quantum population Q(t), at any instant t of the evolutionary process, is formed by a set of N quantum individuals qi (i = 1, 2, 3, .., N). Each quantum individual qi of this population is formed by L genes gij (j = 1, 2, 3, ..., L). The main difference between the QIEA-BR and its predecessors is that part of the L genes is represented by q-bit, similar to QIEA-B, and another part by real quantum genes (q-real, similar to QIEA-R). Thus, the representation of a quantum individual i at any time instant t is given by:

$$q_i = \left[(q_i)_b (q_i)_r\right] \quad (4)$$

where the index b represents the binary part (q-bit) and the index r represents the real part (q-real). Thus a quantum individual can

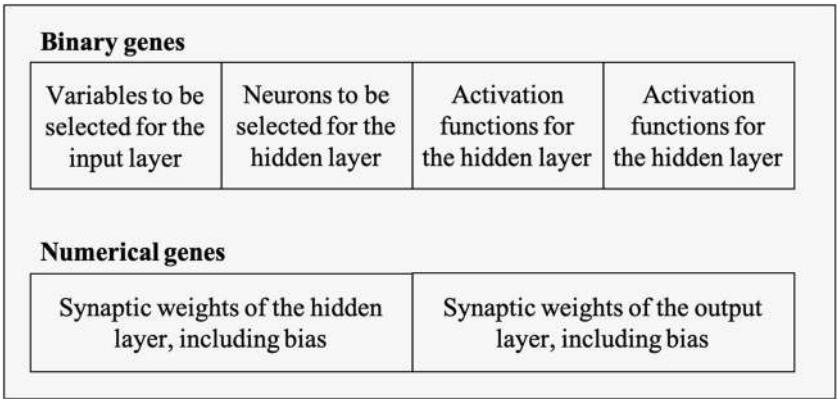

Fig. 1 - The QIEA-BR individual structure [40]

be described by:

$$q_i = [(q_i)_b (q_i)_r] = \left(\left| \begin{matrix} \alpha_{i1} \\ \beta_{i1} \end{matrix} \right\| \left. \begin{matrix} \alpha_{i2} \\ \beta_{i2} \end{matrix} \right| \ldots \left| \begin{matrix} \alpha_{iM} \\ \beta_{iM} \end{matrix} \right| \right)_b \left(\left| \begin{matrix} \mu_{i1} \\ \sigma_{i1} \end{matrix} \right\| \left. \begin{matrix} \mu_{i2} \\ \sigma_{i2} \end{matrix} \right| \ldots \left| \begin{matrix} \mu_{iG} \\ \sigma_{iG} \end{matrix} \right| \right)_r \quad (5)$$

In this work, the QIEA-BR is used to perform the complete modeling of an artificial MLP neural network. The binary part selects the most appropriate input variables; defines which neurons (of a maximum number of neurons) are active in the hidden layer (1 active neuron, 0 inactive); and specifies the activation function of each neuron in the network (1 hyperbolic tangent and 0 sigmoid). The real part determines the values of all weights. Figure 1 illustrates the information that is encoded in each of the quantum genes, binary or real, of a QIEA-BR chromosome. This chromosome will be used in the neuroevolutionary models presented in Section 4.

In QIEA-BR, the evolution of the weights and activation function of a certain neuron in the quantum and classical chromosomes is conditioned to that neuron being active in the corresponding binary part. That is, the genes representing the weights and activation functions will remain unchanged by quantum and classical evolutionary process if this neuron is inactive.

The neural network created by QIEA-BR is similar to that shown in Fig. 2: the effective number of attributes in the input layer and of neurons in the hidden layer are evolved by the QIEA-BR, with the maximum size of inputs equal to the number of available attributes in the dataset (k) and the maximum number of neurons in the hidden layer (nh) configured by the user.

Thus, the number of genes is given by:

$$num_{genes} = (k + 2nh + nc)_b + ((k+1) \times nh) + ((nh+1) \times nc)_r \quad (6)$$

where nc is the number of classes in the classification problem. In this case, the evaluation function used is the classification accuracy given by:

$$Accuracy = 1 - \frac{1}{n} \sum_i |C_i - \hat{C}_i| \quad (7)$$

where C_i is the class of the i-th pattern, while $\hat{C}_i$ is the class predicted by the individual (MLP). When $C_i = \hat{C}_i$ then the result is zero, otherwise it is equal to one. Each individual is submitted to this evaluation function, in such a way that the best individuals are those who have greater accuracy. Further details on QIEA-BR can be found in [40, 52].

3 NEVE: Neuroevolutionary Model for Learning in Nonstationary Environments

This section presents the proposed new quantum-inspired neuroevolutionary model, which is a self-adaptive and flexible

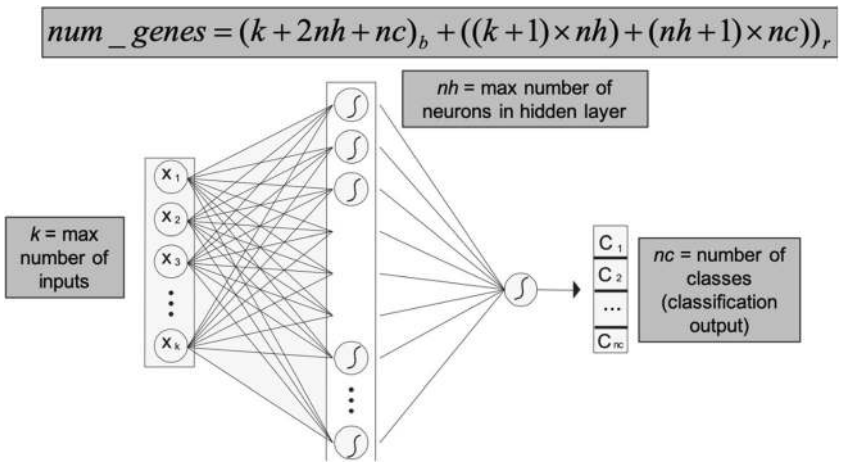

Fig. 2 - Neural Network Created by QIEA-BR

model, with good accuracy and suitable for learning in non-stationary environments. The model is based on an ensemble of neural networks Multi-Layer Perceptron (MLP), where each neural network member is trained and has its parameters (topology, weights, among others) optimized by QIEA-BR algorithm (see Section 2). This neuroevolutionary model is called NEVE (Neuroevolutionary Ensemble) and is composed of three main modules, detailed below and illustrated in Fig. 3:

- Drift Detection;
- Classifier Creation;
- Evaluation and combination weights.

The Drift Detection module is optional. If activated, for each new input data block received, the detection module checks if any drift has occurred. The model works with data blocks of configurable size. If it is necessary (or desired) to work with individual data inputs, the block can be set to size to 1. However, it is important to mention that the strategy of working with one instance at a time is not the most suitable for this model, as it may compromise its computational performance. Two methods of detection were proposed: proactive and reactive detection methods, resulting in four different approaches implemented for this drift detection module [10]:

- **No detection;**
- **Reactive detection:** waits until the real data block labels are available to check if a drift has occurred in relation to the previous data block;
- **Proactive detection (Group Label approach**): for each new data block received, a clustering algorithm is performed using the centroids of the previous labeled data block as initial centroids. Based on the results of the clustering algorithm, the detection mechanism checks if a drift has occurred in relation to the previous block and, if so, a new MLP is created and trained with the new block and the class labels suggested in the clustering;
- **Proactive detection (Pattern Mean Shift approach):** similar to the Group Label approach, with the difference that when a drift is detected, instead of creating a new MLP with the new data block, the old data block is used to train the MLP and the drift is "removed" from the new data block. While in the Group Label approach the new MLP is adjusted to the new data, in Pattern Mean Shift approach the new data is adjusted to the old MLP.

The Classifier Creation Module is responsible for creating a new classifier, which may or may not be added to the ensemble, depending on its maximum size defined by the user. It is worth mentioning that the decision to create a new neural network is linked to the drift detection mechanism used, which will be better detailed in the following subsections. If created, the new classifier is added to the ensemble if space is available or by replacing an older classifier of worse accuracy. This approach gives the ensemble the ability to learn the new data without having to analyze the old data, as well as allowing to forget the data that is no longer needed. In short, the classifier creation module determines the complete configuration of the new MLP network ensemble member using the QIEA-BR algorithm (presented in Section 2). The algorithm selects the most relevant input variables, specifies the number of neurons in the hidden layer (respecting the maximum limit configured by the user), and determines the weights and activation functions of each neuron. The number of output neurons is equal to the number classes in the application.

Finally, the Evaluation Module is responsible for determining the final response of the classifier ensemble by combining the results presented by the classifier members. The QIEA-R algorithm is used to determine the most suitable voting weight

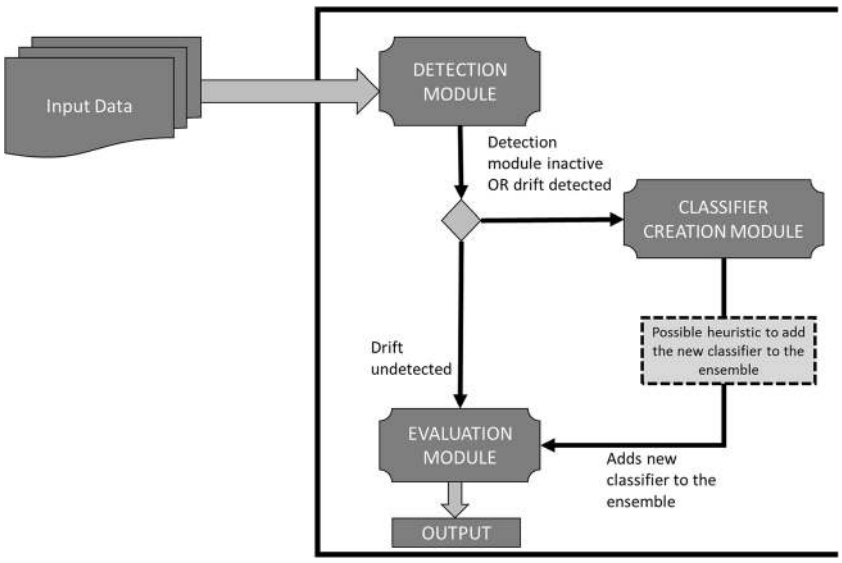

Fig. 3 - Modular structure of NEVE

for each classifier dynamically. The optimization of weights allows the model to easily adapt to sudden data changes by assigning higher weights to the classifiers best suited to the current concepts that govern the data. Three possible voting methods were implemented:

- **Linear Combination:** It uses the QIEA-R algorithm to generate a voting weight for each classifier, which is multiplied by the output of each ensemble member (between 0 and 1), on a weighted average. The result of this weighted average is used to determine the ensemble response. If the problem has only two classes, the output is assigned to class 0 if the result is less than 0.5 and to class 1 otherwise; in case of problems with multiple classes, the class will be the one that presents the output with the highest value;
- **Weighted Majority Voting:** As in the previous case, it uses the QIEA-R algorithm to generate a voting weight for each classifier. However, the outputs of the neurons from each ensemble network are first rounded (for values 0 or 1) and then multiplied by the corresponding classifier weight, thus forming a weighted average. Similar to the linear combination, in problems with only two classes, the output is defined as class 0 if the result of the weighted average is less than 0.5 and as class 1 otherwise; in the case of problems with multiple classes, the class associated with the output with the highest value is defined;
- **Simple Majority Voting:** The output of each ensemble member is rounded to one of the possible classes, and the ensemble final output is the most chosen class among all classifiers. In this case, there is no need to determine voting weights.

In summary, considering the detection mechanism used, there are four possible variations of the NEVE model proposed and detailed in the following subsections:

- ND-NEVE, without detection
- RD-NEVE, with reactive detection
- PDGL-NEVE, with proactive detection and the Group Label approach
- PDPMS- NEVE, with proactive detection and the Pattern Mean Shift approach

The following subsections detail each of the four proposed NEVE variations. For each variation, an explanatory text and a pseudocode of the algorithm is presented.

3.1 ND-NEVE (without detection)

The first variation of NEVE, "NEVE without Detection" (ND-NEVE), as the name implies, does not use any detection mechanism. It consists of an ensemble of MLP neural networks that, with each new data block received, it trains a new MLP that can be added to the ensemble if space is available.

The operation of ND-NEVE can be generalized as: when a data block t arrives (without the class labels), a new MLP network is trained using the QIEA-BR algorithm and t-1 block with the real class labels. The new network is provisionally added to the ensemble and the ensemble is tested with block t. Voting weights of all networks are determined using the QIEA-R algorithm and block t-1. The final ensemble classification is calculated using the test results with block t, the voting weights and the chosen voting method. Finally, we assume that the actual labels of block t become available and then, the permanence of the new network in the ensemble is evaluated. The pseudocode of the ND-NEVE is demonstrated in Appendix 2.

3.2 RD-NEVE (with reactive detection)

The second variation of NEVE is "RD-NEVE (with reactive detection)". This variation uses the reactive detection mechanism, detailed in [10]. For each new data block received, the ensemble classifies it and, as soon as the real class labels are obtained, the detection mechanism checks if a drift has occurred from the previous data block. If so, a new MLP is created, which is added to the ensemble if space is available.

The operation of RD-NEVE can be generalized as follows: when a data block t arrives, the voting weights of all ensemble members are determined using the QIEA-R algorithm and the t-1 block. The ensemble is tested with block t and classification results are combined with the weights calculated by QIEA-R, using the chosen voting method to determine the final ensemble classification. It is assumed that the real labels of block t are later available and the reactive detection can be applied [10]. If a drift has occurred in block t, a new MLP network is created using the QIEA-BR algorithm and trained with block t. The new network is added to the ensemble if space is available or if it is better than at least one of the old networks, replacing it on the ensemble. The pseudocode of the DE-NEVE is demonstrated in Appendix 2.

3.3 PDGL-NEVE (with proactive detection and Group Label approach)

The third variation of NEVE is "PDGL-NEVE (with proactive detection and Group Label approach)". This variation uses the proactive mechanism of detection [10], where each new data block is clustered, using the centroids of the previous data block as the initial centroids of the algorithm. Based on the clustering results, the detection mechanism checks if a drift has occurred from the previous data block; if so, the model trains a new MLP with the new data block and the class labels suggested by the clustering algorithm.

The operation of PDGL-NEVE can be summarized as: when block t arrives, its instances are grouped using the real classes of block t-1 as the initial suggestion of centroids, since the real class labels of block t are still unknown. Then, it is verified if a drift has occurred in block t in relation to block t-1. If so, a new MLP network is created using the QIEA-BR algorithm and trained using block t with the class labels provided by the clustering algorithm. The new network is provisionally added to the ensemble, which is tested with block t. The voting weights for all networks are determined using the QIEA-R algorithm and block t, also with the labels provided the clustering algorithm. The classification results and weights are combined using the chosen voting method to determine the final ensemble classification. It is assumed that the real labels of block t are later available and the initial centroids for the next grouping are updated, now considering the real class labels of the data block. The permanency of the new network in the ensemble is evaluated: it stays if space is available or if it is better than at least one of the old networks, replacing it in the ensemble. The pseudocode of the PDGL-NEVE is demonstrated in Appendix 2.

3.4 PDPMS-NEVE (with proactive detection and Pattern Mean Shift approach)

The fourth variation of the NEVE is "PDPMS-NEVE (with proactive detection and Pattern Mean Shift approach)". This variation also uses the proactive detection [10]. As in the previous variation, each new data block is grouped to verify if a drift has occurred from the previous data block. If so, a new MLP is trained with the previous labeled data block, and the new data block is "adjusted" towards the previous data block. In other words, when a drift is detected, instead of creating a new MLP using the new data block (as performed by the Group Label approach), the old data block is used to train the network and the drift is "removed" from the new data block. While in the Group Label approach the new network is suitable for the new data, in Pattern Mean Shift approach the new data is adjusted to the old network (trained with the old data). The pseudocode of the PDPMS-NEVE is demonstrated in Appendix 2.

Briefly, the main difference between PDGL-NEVE and PDPMS-NEVE is that in PDPMS-NEVE, when a drift is detected, a new MLP is created using the previous labeled data block (and not the new data block with the labels provided in the grouping, as in the PDGL-NEVE). Then, the new data block is "adjusted" in the direction of the previous data block and it is submitted to the ensemble classification. In the PDGL-NEVE, on the other hand, the new data block is tested by the ensemble without adjusting the data. Additionally, in PDPMS-NEVE the data block used to determine the weights of each MLP is the old data block with the real labels, while in the PDGL-NEVE the new data block is used with the labels provided by the grouping.

This section presented the neuroevolutionary model for learning in nonstationary environments proposed in this paper and detailed its four variations. The next section describes the experiments performed with the proposed detection methods.

4 Experiments

To assess the ability of the proposed model to learn in nonstationary environments and also to verify the best variations and configurations of the models regarding accuracy and computational performance, six different datasets were used on different simulations and scenarios. For the experiments, the four variations of the proposed model (described in Section 3) were used: ND-NEVE, RD-NEVE, PDGL-NEVE and PDPMS-NEVE. All experiments were run using standard libraries of MATLAB, as well as its Neural Networks package to train the baseline networks.

4.1 Datasets description

The datasets used in the experiments are: the SEA Concepts (an artificial dataset with a more controlled environment about the drifts) and four real datasets (Nebraska, Electricity, Cover Type and Poker Hand), where the exact moment that the drift occurs is unknown.

The SEA Concepts dataset was artificially created by [49]. It is characterized by extensive periods without major changes in the environment, but with occasional abrupt drifts. The Nebraska dataset presents a compilation of climate measurements from the Offutt Air Force Base substation in Bellevue, Nebraska. Its objective is to predict whether a rainfall may appear, using data from the last 30 days. Both datasets are available in [41]. The Electricity dataset is extracted from the Australian New South Wales Electricity Market and the class label defines the price change related to a moving average of the last 24 h. The purpose of the problem is to predict whether the price will go up or down. The Cover Type dataset contains information cells corresponding to a forest cover of 30×30 meters, extracted from the US Forest Service (USFS). Its goal is to predict the type of forest cover among seven possible values (therefore, a multi-class problem). The Poker Hand dataset has ten possible categories as output, representing the poker hand that contains 5 cards. The purpose is to identify the type of a Poker hand among the ten possibilities. These datasets are available in [34]. Table 2 presents the main features of each dataset, as well as the block size and number of blocks used in the experiments.

Table 2 – Datasets Details

Dataset	Block Size	Number of Blocks	Number of Inputs	Number of Classes
SEA Concepts	250	400	3	2
Nebraska	30	583	8	2
Electricity	48	944	8	2
Cover Type	500	1162	54	7
Poker Hand	500	1658	10	10

4.2 Execution details

All executions begin at t = 0 and end when consecutive T data blocks are presented for training and testing, with each block being able to suffer different scenarios of concept drift with unknown rates and natures.

As detailed in Section 3, the QIEA-BR algorithm evolves the topology of each new neural network, which is created following the criteria of each variation of the proposed model. The number of input variables is selected by QIEA-BR among the available variables in each dataset. For all datasets, a single hidden layer was used, whose number of neurons is evolved by QIEA-BR, having a maximum value specified by the user. The number of neurons in the output layer is equal to the number of classes in each dataset. The synaptic weights and activation functions of the hidden layer and the output layer are also determined by QIEA-BR.

The parameters of the quantum evolutionary algorithms are the same as those used by [1, 40] and they are detailed in Table 3. The three voting methods detailed in Section 3 were evaluated: linear combination, weighted majority voting and simple majority voting. The maximum ensemble size is also a parameter defined by the user. Table 3 presents the configuration of the parameters used in all the experiments.

Thus, for each dataset, 72 different configurations of the model ($4 \times 3 \times 3 \times 2$) were used, representing each possible combination of the parameters to be evaluated, as shown in Table 3. For each configuration, 30 simulations were performed and the average accuracy and computational time of these runs were calculated.

5 Results

The experiments presented below aimed at investigating the difference between accuracy (the ratio of number of correct predictions to the total number of input samples) and computational performance (execution time in seconds) among each of the four variations of the NEVE model, as well as the impact of the voting method, ensemble size and number of neurons in the hidden layer. Therefore, the objective of the experiment is to analyze how these modifications affect the results of the models for each dataset.

Tables 4, 5, 6 and 7 show the results of the experiments performed considering the accuracy and the computational performance measured in seconds. It should be noted that execution time is provided only for the SEA Concepts, Nebraska and Electricity datasets. Due to the considerable size of

Table 3 – Experiments Settings

Parameter	Possible Values
Model variation	ND-NEVE (without detection)
	RD-NEVE (reactive detection)
	PDGL-NEVE (proactive detection, Group Label)
	PDPMS-NEVE (proactive detection, Pattern Mean Shift
Voting method	Linear Combination
	Weighted Majority Voting
	Simple Majority Voting
Ensemble size	5
	10
	Unlimited
Maximum Number of neurons in the hidden layer	5
	10
Quantum evolutionary algorithms parameters	Lambda (penalization parameter) = 0.001
	Quantum population size = 10
	Classical population size = 20
	Number of generations = 100
	Quantical and classical crossover rate = 0.1
	Number of generations before checking improvement = 4

Table 4 – Results for Dataset SEA Concepts

SEA	Accuracy						Time in seconds					
Hidden	5	10	5	10	5	10	5	10	5	10	5	10
Ensemble	5	5	10	10	Un	Un	5	5	10	10	Un	Un
	ND-NEVE						ND-NEVE					
lin. comb.	**0,91**	**0,91**	**0,91**	**0,90**	0,87	0,87	**270**	311	356	364	*527*	*549*
weigh. maj	**0,90**	**0,90**	0,89	0,89	0,83	0,85	438	401	421	424	*590*	*603*
simpl. maj	**0,90**	**0,90**	0,90	0,90	0,86	0,86	378	367	387	378	486	521
	RD-NEVE						RD-NEVE					
lin. comb.	**0,90**	**0,90**	**0,90**	**0,91**	0,87	0,87	**289**	328	374	379	*536*	*568*
weigh. maj	**0,90**	**0,90**	0,90	0,88	0,86	0,86	444	408	429	429	*611*	*599*
simpl. maj	0,90	0,90	0,90	0,90	0,86	0,86	**279**	**274**	**282**	**280**	292	**289**
	PDGL-NEVE						PDGL-NEVE					
lin. comb.	*0,72*	*0,73*	*0,73*	*0,74*	*0,75*	*0,75*	364	377	382	374	469	459
weigh. maj	*0,69*	*0,69*	*0,68*	*0,68*	*0,59*	*0,59*	403	414	447	443	*607*	*605*
simpl. maj	0,76	0,76	0,77	0,77	0,81	0,80	**288**	**288**	**290**	**290**	307	310
	PDPMS-NEVE						PDPMS-NEVE					
lin. comb.	0,87	0,87	0,87	0,87	0,85	0,85	**276**	334	366	394	*568*	*567*
weigh. maj	0,87	0,88	0,86	0,86	*0,61*	*0,64*	422	411	438	444	*612*	*598*
simpl. maj	0,87	0,87	0,87	0,87	0,85	0,85	**288**	296	297	**287**	302	598

Poker Hand and Cover Type datasets, their execution required the parallelization on several computers, making the comparison of runtime between simulations impracticable. In all cases, the observed standard deviation was less than 2%. We highlighted the best 20% results in bold and gray and the worst 20% results in italics and underlined.

The analysis of Tables 4 to 7 shows that:

- In general, the ND-NEVE, RD-NEVE and PDPMS-NEVE approaches provided the best accuracy, while the PDGL-NEVE had the worst accuracy;
- Considering computational performance, the ND-NEVE, RD-NEVE and PDPMS-NEVE approaches presented the best computational times, and the PDGL-NEVE approach, the worst. It was observed, however, that the

Table 5 – Results for Dataset Nebraska

Nebraska	Accuracy						Time in seconds					
Hidden	5	10	5	10	5	10	5	10	5	10	5	10
Ensemble	5	5	10	10	Un	Un	5	5	10	10	Un	Un
	ND-NEVE						ND-NEVE					
lin. comb.	0,69	0,69	**0,70**	**0,70**	**0,70**	**0,70**	1052	1047	1030	1041	*1277*	*1283*
weigh. maj	0,67	0,68	0,68	0,67	**0,70**	0,70	1169	1161	1170	1178	*1668*	*1630*
simpl. maj	0,69	0,70	0,70	0,69	**0,70**	**0,70**	1058	1056	951	863	1196	1231
	RD-NEVE						RD-NEVE					
lin. comb.	0,69	0,69	**0,70**	**0,70**	**0,70**	**0,70**	1022	1025	1017	1008	1252	1253
weigh. maj	0,68	0,67	0,68	0,68	0,69	**0,70**	1124	1127	1122	1125	*1620*	*1620*
simpl. maj	0,69	0,70	**0,70**	0,70	0,70	**0,70**	800	**792**	**794**	**737**	**753**	**749**
	PDGL-NEVE						PDGL-NEVE					
lin. comb.	*0,53*	*0,53*	*0,53*	*0,53*	0,55	0,55	1020	1024	1022	1037	*1266*	*1261*
weigh. maj	*0,48*	*0,48*	*0,43*	*0,44*	*0,33*	*0,34*	1137	1145	1154	1140	*1594*	*1437*
simpl. maj	0,55	0,55	*0,49*	*0,50*	0,56	0,57	**660**	**668**	**688**	**689**	**785**	**786**
	PDPMS-NEVE						PDPMS-NEVE					
lin. comb.	0,65	0,65	0,67	0,66	0,68	0,68	1047	1055	1034	1034	*1262*	*1261*
weigh. maj	0,63	0,63	0,63	0,63	*0,36*	*0,37*	1149	1147	1157	1165	*1592*	*1591*
simpl. maj	0,67	0,67	0,67	0,67	0,69	0,69	824	816	817	**758**	**768**	**761**

Table 6 – Results for Dataset Electricity

Electricity	Accuracy						Time in seconds					
Hidden	5	10	5	10	5	10	5	10	5	10	5	10
Ensemble	5	5	10	10	Un	Un	5	5	10	10	Un	Un
	ND-NEVE						**ND-NEVE**					
lin. comb.	0,54	0,54	0,54	0,55	0,56	*0,50*	1428	1451	1523	1645	*2907*	2490
weigh. maj	0,55	0,55	0,60	0,59	0,62	0,62	1673	1660	1694	1691	*2947*	*2754*
simpl. maj	0,55	0,53	0,55	0,56	*0,44*	*0,49*	1481	1444	1423	1486	2495	*2592*
	RD-NEVE						**RD-NEVE**					
lin. comb.	0,53	0,53	0,54	0,55	*0,50*	*0,52*	1366	1376	1389	1393	2431	*2599*
weigh. maj	0,56	0,56	0,59	0,59	0,63	0,62	1671	1629	1580	1592	*2793*	*2829*
simpl. maj	*0,52*	*0,52*	0,55	0,53	*0,45*	*0,45*	**1124**	**1116**	**1101**	**1089**	1384	1416
	PDGL-NEVE						**PDGL-NEVE**					
lin. comb.	**0,71**	**0,71**	**0,71**	**0,71**	**0,72**	**0,72**	1391	1390	1443	1456	*2506*	*2513*
weigh. maj	**0,71**	**0,72**	**0,71**	**0,72**	0,66	0,66	1614	1543	1557	1579	*2712*	*2604*
simpl. maj	0,69	0,68	**0,70**	**0,70**	**0,71**	**0,69**	**1049**	**1036**	**1031**	**1050**	1335	1348
	PDPMS-NEVE						**PDPMS-NEVE**					
lin. comb.	0,53	0,53	0,53	0,54	0,54	*0,46*	1382	1363	1347	1352	2308	*2626*
weigh. maj	0,55	0,55	0,57	0,58	0,58	0,58	1703	1615	1562	1561	*2566*	*2601*
simpl. maj	*0,52*	*0,52*	0,54	0,54	*0,44*	*0,46*	**1121**	**1083**	**1077**	**1067**	**1297**	**1282**

dataset also has a great influence on this criterion: the slowest was Electricity, which is the dataset that has the highest number of attributes and also a greater number of blocks among the datasets evaluated;

– The best voting methods in terms of accuracy are, in this order: linear combination, followed by weighted majority and simple majority. This shows that the quantum algorithm is contributing positively to the accuracy of the model by determining the voting weights of the networks. Possibly, the early rounding performed in the weighted majority resulted in in attaining a lower average accuracy than the linear combination;
– As for the computational performance, the best voting method was the simple majority, which was already expected since this method does not perform the determination of weights via quantum algorithm;

Table 7 – Results for Datasets Poker Hand e Cover Type

	Poker Hand - Accuracy						Cover Type - Accuracy					
Hidden	5	10	5	10	5	10	5	10	5	10	5	10
Ensemble	5	5	10	10	Un	Un	5	5	10	10	Un	Un
	ND-NEVE						**ND-NEVE**					
lin. comb.	0,59	0,60	**0,61**	0,60	**0,63**	**0,63**	**0,74**	**0,73**	**0,75**	**0,73**	0,58	0,53
weigh. maj	0,59	0,59	**0,60**	**0,60**	**0,64**	**0,63**	**0,75**	**0,74**	**0,73**	0,71	0,59	0,58
simpl. maj	0,56	0,56	0,54	0,54	0,53	0,54	0,69	0,70	0,70	0,71	*0,50*	*0,49*
	RD-NEVE						**RD-NEVE**					
lin. comb.	0,59	0,60	**0,61**	0,60	**0,63**	**0,63**	**0,74**	**0,73**	**0,75**	**0,73**	0,59	0,54
weigh. maj	0,59	0,58	**0,61**	**0,60**	**0,64**	**0,64**	**0,75**	**0,75**	**0,74**	**0,73**	0,61	0,56
simpl. maj	0,57	0,56	0,54	0,54	0,54	0,54	0,69	0,71	0,70	0,69	*0,51*	0,52
	PDGL-NEVE						**PDGL-NEVE**					
lin. comb.	*0,25*	*0,25*	*0,37*	*0,38*	*0,41*	*0,35*	0,51	*0,51*	0,54	0,54	*0,47*	*0,49*
weigh. maj	*0,23*	*0,22*	*0,41*	*0,41*	*0,38*	*0,40*	*0,50*	*0,49*	0,54	0,55	*0,50*	*0,50*
simpl. maj	0,47	0,47	0,48	0,46	*0,40*	*0,41*	0,56	0,57	0,55	0,56	*0,46*	*0,42*
	PDPMS-NEVE						**PDPMS-NEVE**					
lin. comb.	0,58	0,57	0,56	0,57	0,58	0,57	0,72	0,70	0,70	0,72	0,59	0,53
weigh. maj	0,57	0,57	0,56	0,57	0,59	0,59	0,72	0,72	0,70	0,71	0,56	0,55
simpl. maj	0,55	0,55	0,53	0,52	0,51	0,53	0,71	0,70	0,70	0,69	*0,49*	*0,49*

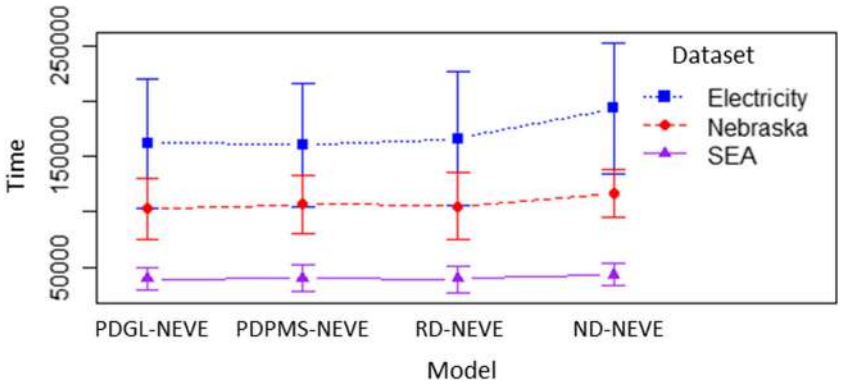

Fig. 4 - Comparative runtime analysis

- It is observed that, in general, the strategy of unlimited ensemble has lower accuracy than the limited ensembles. There was no significant difference in accuracy between the 5 and 10 ensemble size, which is a positive point, because the unlimited ensemble strategies also presented the worst computational performance, as expected. The unlimited ensemble tends to provide worse accuracy probably due to the increase in the search space of the QIEA-R for determining the voting weights when there are too many networks: it is enough to observe that, in all the datasets used, there are at least 400 data blocks, which allows ensembles of 400 networks for the unlimited case;
- No substantial differences were observed either in the average accuracy or in the average computational performance considering the strategies of 5 and 10 neurons maximum in the hidden layer.

Figure 4 presents a comparative graph of the computational time for the three binary datasets: SEA, Nebraska and Electricity datasets. It can be observed that the computational time of the ND-NEVE approach is superior to the others, whereas approaches with some type of detection present a similar mean computational time. This confirms that the proposed detection mechanism contributes to reducing the average execution time of the models.

The accuracy of the proposed NEVE approaches was also compared with DWM [26], Learn ++, NSE [9], RCD [16], EFPT [55] and AMANDA [56] models. We used 3 different drift detectors for the RCD algorithm: DDM [14], EDDM [5] and ECDD [42]. These simulations were carried out using MOA [35], an open source framework for data mining that includes several learning algorithms implemented for classification, regression, clustering, concept drift detection, among others. For this comparison, we used the same block size chosen for NEVE simulations for all the datasets. In order to make a more coherent comparison with NEVE and to discard the influence of the base classifier on the accuracy of the model, in all other models, the MLP neural networks were used as base classifiers. All the models were parameterized using values indicated by the respective authors.

Table 8 presents the results of the best reached configuration (in terms of accuracy) of each NEVE variation, compared to the results of the other models. We highlighted the best results, by dataset, in bold and underlined, the second best in bold and the worst in italics and underlined. When more than one value is highlighted, it means that there is no statistically significant difference in the performance of the classifiers for ≤ 0.05, according to Wilcoxon test. We made 30 runs for each possible configuration and each dataset. In all cases, the observed standard deviation was less than 2%.

We can see from Table 8 that NEVE approaches obtained the best result in 2 datasets and the second best in the other 3. Apparently, the ND-NEVE and RD-NEVE approaches provide uniformly superior results in terms of accuracy. What is noticeable in this experiment, in general, is that the EFPT model is the main competitor of the NEVE in terms of accuracy considering SEA, Nebraska and Electricity datasets (as the author didn't performed tests with Poker and Covtype datasets, we

Table 8 – Comparison of results: Best case of NEVE x other models

Best	ND-NEVE	RD-NEVE	PDGCL-NEVE	PDPMS-NEVE	RCD (DDM)	RCD (EDDM)	RCD (ECDD)	DWM	Learn^{++}.NSE	EFPT	Amanda
SEA	**90.53**	**90.53**	81,08	**87,56**	82,50	80,22	80,37	60,67	35.31	**91.60**	N/A
Nebraska	**70,13**	**70,20**	56,74	**68,76**	40,28	46,16	64,00	50,08	30.05	**71,11**	67,90
Electricity	62,03	63,04	**71,68**	58,04	52,41	48,22	49,41	67,16	42.46	**77,90**	69,00
Poker	**64,34**	63,77	47,52	59,02	60,37	59,30	40.44	**73,78**	47,76	N/A	N/A
Covtype	**75,30**	74,83	57,04	72,48	46,34	33,78	57,19	**81,91**	28.05	N/A	N/A

could not compare the models in this datasets) and the DWM models seems to be the main competitor of the NEVE in terms of accuracy considering Poker and Covtype datasets.

From the results presented, we can highlight that NEVE provides good results without the need for a detection method; however, by adding one, substantial gains in accuracy and computational performance can be obtained. This fact reinforces that the neuroevolutionary ensemble approach is a robust choice for situations in which datasets are subject to sudden behavioral changes.

6 Conclusion

This work presented a new neuroevolutive model with quantum-inspiration, based on a multi-layer perceptron (MLP) neural network ensemble for learning in nonstationary environments, called NEVE (Neuro-EVolutionary Ensemble). This model can be used in conjunction with the DetectA concept drift detection model [10], which has the ability to detect changes both proactively and reactively. The use of Quantum-Inspired Evolutionary Algorithms in conjunction with NEVE allows the automatic generation of new classifiers for the ensemble (including the decision of its topology, the most appropriate input variables and its weights) and determining the voting weights of each neural network member of the ensemble.

Four different variations of NEVE were implemented: ND-NEVE (without detection), RD-NEVE (with reactive detection), PDGL-NEVE (with proactive detection and Group Label approach), PDPMS-NEVE (with proactive detection and Pattern Mean Shift approach). These variations differ from each other in the way they detect and treat drifts, and were used in experiments with real and artificial datasets in order to evaluate which model variation and configurations achieved the best results. We varied the voting method, the maximum number of neurons in the hidden layer and the maximum size of the ensemble. It was found that the ND-NEVE, RD-NEVE and PDPMS-NEVE approaches produce best results in terms of accuracy and computational performance. It was also observed that the linear combination is the best voting method in terms of accuracy, and simple majority voting the best in terms of computational performance. The unlimited ensemble strategy has worse accuracy and computational performance than limited ensembles, with no significant difference between the 5 and 10 networks.

Compared with other consolidated models of the literature, the accuracy of NEVE was found to be superior in most cases. It appeared that the ND-NEVE and RD-NEVE approaches provide uniformly superior results in terms of accuracy, but the addition of the detection method in some cases has resulted in substantial gains. This fact reinforces that the neuroevolutionary ensemble approach was a robust choice for situations in which datasets are subject to sudden behavioral changes.

As future work, we intend to integrate, in a single evolutionary model, the creation of the neural network and the determination of voting weights, in order to perform the evolution process in a single integrated process. Also, it is intended to use NEVE for real applications, in order to validate its practical use, although it is very hard to know for sure if a dataset contains concept drift or not.

Appendix 1 – Pseudocode of the QIEA-R algorithm

The pseudocode of the QIEA-R algorithm is shown as follows.

```
1. t <- 1
2. Create quantum pop. Q(t) with N individuals with G genes
3. while (t <= T)
3.1.  Create the PDF's for the quantum individuals
3.2.  E(t) <- generate classical individuals observing quantum
      individuals
3.3.  if (t=1) then
3.3.1. C(t) <- E(t)
3.4.  else
3.4.1. E(t) <- Crossover between E(t) and C(t)
3.4.2. Evaluate E(t)
3.4.3. C(t) <- K best individuals from [E(t) + C(t)]
3.5.  Q(t+1) <- update Q(t) using N best individuals from C(t)
3.6.  t <- t+1
```

Appendix 2 – Pseudocode of NEVE algorithms

The pseudocode of the ND-NEVE is demonstrated as follows.

```
   Create an empty ensemble of classifiers P
1. Set s as the maximum ensemble size
2. Create a new MLP classifier c' using the data block D1 and QIEA-BR
   and add it to the ensemble P
3. Test c' with the data block D2
4. Ensemble final decision = c' decision
5. Receive real labels of D2
6. For each data block Di and i = 3, 4..., m do
6.1.  Create a new MLP classifier and train it using data block Di-1
      and QIEA-BR
6.2.  Add the new classifier provisionally to the ensemble
6.3.  Test each ensemble classifier with the data block Di
6.4.  Evolve voting weights wj for each classifier using the last data
      block Di-1 and QIEA-R
6.5.  Determine the ensemble final decision using the chosen voting
      method
6.6.  Receive real labels of Di
6.7.  Calculate the classification error E' for the new classifier c'
      and Ej for each ensemble classifier cj and data block Di
6.8.  Calculate the ensemble classification error
6.9.  If ensemble is full (number of classifiers = s) then
i)    If (E' < max(Ej))
   a. Replace classifier with max(Ej) by the new classifier
```

The pseudocode of the DE-NEVE is shown as follows.

```
   Create an empty ensemble of classifiers P
1. Set s as the maximum ensemble size
2. Create a new MLP classifier c' using the data block D1 and QIEA-BR
   and add it to the ensemble P
3. Test c' with the data block D2
4. Ensemble final decision = c' decision
5. Receive real labels of D2
6. If a drift is detected between data blocks D1 and D2
6.1.  Create a new MLP classifier and train it using data block D2 and
      QIEA-BR
7. For each data block Di and i = 3, 4..., m do
7.1.  Evolve voting weights wj for each classifier using the last data
      block Di-1 and QIEA-R
7.2.  Test each ensemble classifier with the data block Di
7.3.  Determine the ensemble final decision using the chosen voting
      method
7.4.  Receive real labels of Di
7.5.  If a drift is detected between data blocks Di-1 and Di
7.6.  Create a new MLP classifier and train it using data block Di and
      QIEA-BR
7.7.  Add the new classifier provisionally to the ensemble
7.8.  Calculate the classification error E' for the new classifier c'
      and Ej for each ensemble classifier cj and data block Di
6.10. If ensemble is full (number of classifiers = s) then
   i)    If (E' < max(Ej))
      a. Replace classifier with max(Ej) by the new classifier
7.9.  Calculate the ensemble classification error
```

The pseudocode of the PDGL-NEVE is demonstrated as follows:

```
   Create an empty ensemble of classifiers P
1. Set s as the maximum ensemble size
2. Create a new MLP classifier c' using the data block D1 and QIEA-BR
   and add it to the ensemble P
3. For each data block Di and i = 2, 3..., m do
3.1.  Group Di using Di-1 classes as suggestions of centroids
3.2.  If a drift is detected between data blocks Di-1 and Di
3.2.1.Create a new MLP classifier and train it using data block Di
      with labels provided in clustering and QIEA-BR
3.2.2.Add the new classifier provisionally to the ensemble
3.3.  Test each ensemble classifier with the data block Di
3.4.  Evolve voting weights wj for each classifier using the data
      block Di with labels provided in clustering and QIEA-R
3.5.  Determine the ensemble final decision using the chosen voting
      method
3.6.  Receive real labels of Di
3.7.  Calculate the classification error E' for the new classifier c'
      and Ej for each ensemble classifier cj and data block Di
3.8.  Update the suggestions of centroids for next clustering using
      data block Di and the real labels
3.9.  If a new classifier was created and if ensemble is full (number
      of classifiers = s) then
   i)    If (E' < max(Ej))
      a. Replace classifier with max(Ej) by the new classifier
```

The pseudocode of the PDPMS-NEVE is demonstrated as follows:

```
   Create an empty ensemble of classifiers P
1. Set s as the maximum ensemble size
2. Create a new MLP classifier c' using the data block D1 and QIEA-BR
   and add it to the ensemble P
3. For each data block Di and i = 2, 3..., m do
3.1.  Group Di using Di-1 classes as suggestions of centroids
3.2.  If a drift is detected between data blocks Di-1 and Di
3.2.1.Create a new MLP classifier and train it using data block Di-1
      with real labels and QIEA-BR
3.2.2.Add the new classifier provisionally to the ensemble
3.2.3.Adjust data block Di to make it similar to data block Di-1
3.3.  Test each ensemble classifier with the data block Di
3.4.  Evolve voting weights wj for each classifier using the data
      block Di-1 with real labels and QIEA-R
3.5.  Determine the ensemble final decision using the chosen voting
      method
3.6.  Receive real labels of Di
3.7.  Calculate the classification error E' for the new classifier c'
      and Ej for each ensemble classifier cj and data block Di
3.8.  Update the suggestions of centroids for next clustering using
      data block Di and the real labels
3.9.  If a new classifier was created and if ensemble is full (number
      of classifiers = s) then
   i)    If (E' < max(Ej))
      a. Replace classifier with max(Ej) for the new classifier
```

References

1. Abs da Cruz AV (2007) Algoritmos evolutivos com inspiração quântica para otimização de problemas com representação numérica. PhD Thesis, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, (in portuguese)
2. Abs da Cruz AV, Vellasco MMBR, Pacheco MAC (2008) Quantum-inspired evolutionary algorithm for numerical optimization. In Quantum inspired intelligent systems, pp. 115–132. Springer, Berlin Heidelberg
3. Alippi C, Liu D, Zhao D, Bu L (2014) Detecting and Reacting to Changes in Sensing Units: The Active Classifier Case. IEEE Transactions on Systems, Man, and Cybernetics: Systems 44(3): 353–362
4. Bach SH, Maloof MA (2012) Paired Learners for Concept Drift. Proc. of the 8th IEEE Int. Conf. on Data Mining (ICDM). IEEE, 23–32. Charts for Detecting Concept Drift. Pattern Recogn. Lett. 33, 2, pp. 191–198
5. Baena-García M, Del Campo-Ávila J, Fidalgo R, Bifet A (2006) Early drift detection method. Proc. of the 4th ECML PKDD International Workshop on Knowledge Discovery From Data Streams (IWKDDS'06), Berlin, Germany, pp. 77–86
6. Brzezinski D, Stefanowski J (2014) Reacting to different types of concept drift: The accuracy updated ensemble algorithm. IEEE Trans on Neural Netw Learn Syst 25(1):81–94
7. Carvalho V, Cohen W (2006) Single-Pass Online Learning: Performance, Voting Schemes and Online Feature Selection. Proc. of the 12th ACM SIGKDD Int. Conf. on Knowl. Disc. and DataMining (KDD) ACM, pp. 548–553
8. Dias DM, Pacheco MAC (2012) Quantum-inspired linear genetic programming as a knowledge management system. Comput J 56(9):1043–1062
9. Elwell R, Polikar R (2011) Incremental Learning of Concept drift in Nonstationary Environments. IEEE Trans Neural Netw 22(10): 1517–1531
10. Escovedo T, Koshiyama A, Abs da Cruz A, Vellasco M (2017) DetectA: Abrupt Concept Drift Detection in Non-Stationary Environments. Appl Soft Comput (accepted for publication)
11. Fan W (2004) StreamMiner: a classifier ensemble-based engine to mine conceptdrifting data streams. In Proceedings of the 30th International Conference on Very Large Data Bases, pp. 1257–1260
12. Fan W (2004) Systematic data selection to mine concept-drifting data streams. In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 128–137
13. Frías-Blanco I, del Campo-Avila J, Ramos-Jimenez G, Morales-Bueno R, Ortiz-Díaz A, Caballero-Mota Y (2015) Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. IEEE Transaction On Knowledge Data Engineering 27(3):810–823
14. Gama J, Medas P, Castillo G, Rodrigues PP (2004) Learning with drift detection. Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence, São Luis, Maranhão, Brazil, pp. 286–295
15. Gama J, Žliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Computing Surveys (CSUR) 46(4):44
16. Gonçalves Júnior PM (2013) Multivariate Non-Parametric Statistical Tests to Reuse Classifiers in Recurring Concept Drifting Environments. PhD Thesis, Federal University of Pernambuco, Recife
17. Han K, Kim J (2000) Genetic quantum algorithm and its application to combinatorial optimization problem. Proceedings of the 2000 Congress on Evolutionary Computation 2:1354–1360
18. Han K, Kim J (2002) Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans Evolutionary Computation 6(6):580–593
19. Han K, Kim J (2003) On setting the parameters of qea for practical applications: Some guidelines based on empirical evidence. GECCO:427–428
20. Han K, Kim J (2004) Quantum-inspired evolutionary algorithms with a new termination criterion, He gate, and two-phase scheme. IEEE Trans Evolutionary Computation 8(2):156–169
21. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In Proc. of The 2001 ACM Sigkdd Intl. Conf. on Knowledge Discovery and Data Mining, pp. 97–106
22. Karnick T, Ahiskali M, Muhlbaier M, Polikar R (2008) Learning concept drift in nonstationary environments using an ensemble of classifiers based approach. IJCNN, pp. 3455–3462
23. Khamassi I, Sayed-Mouchaweh M (2014) Drift detection and monitoring in non-stationary environments. Evolving and Adaptive Intelligent Systems (EAIS), 2014 IEEE Conference on, pp. 1–6. IEEE
24. Kolter J, Maloof M (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. Proceedings of the 3rd International IEEE Conference on Data Mining, pp. 123–130
25. Kolter J, Maloof M (2005) Using additive expert ensembles to cope with concept drift. In Proceedings of the 22nd International Conference on Machine Learninig, pp. 449–456
26. Kolter J, Maloof M (2007) Dynamic weighted majority: An ensemble method for drifting concepts. J Mach Learn Res 8:2755–2790
27. Krawczyk B, Minku LL, Gama J, Stefanowski J, Woźniak M (2017) Ensemble learning for data stream analysis: A survey. Information Fusion 37:132–156
28. Kuncheva LI, Faithfull WJ (2014) PCA Feature Extraction for Change Detection in Multidimensional Unlabeled Data. IEEE Transactions on Neural Networks and Learning Systems 25(1):69–80
29. Kuncheva LI (2004) Classifier ensemble for changing environments. in Multiple Classifier Systems, vol. 3077. New York: Springer-Verlag
30. Kuncheva LI (2008) Classifier ensemble for detecting concept change in streaming data: Overview and perspectives. In Proc. Eur. Conf. Artif. Intell, pp. 5–10
31. Maayan H, Mannor S, El-Yaniv R, Crammer K (2014) Concept Drift Detection Through Resampling. In ICML, pp. 1009–1017
32. Minku L, White A, Yao X (2010) The impact of diversity on online ensemble learning in the presence of concept drift. IEEE Trans Knowl Data Eng 22(5):730–742
33. Minku L, Yao X (2012) DDD: A New Ensemble Approach for Dealing With Concept Drift. IEEE Transactions on Knowledge and Data Engineering, IEEE 24(4):619–633
34. MOA Datasets (2018) MOA – Massive Online Analysis. Avaliable at: http://moa.cms.waikato.ac.nz/datasets/
35. MOA (2018) MOA – Massive Online Analysis. Available at: http://moa.cms.waikato.ac.nz/
36. Nishida K, Yamauchi K (2007) Adaptive classifiers-ensemble system for tracking concept drift. In Proceedings of the Sixth International Conference on Machine Learning and Cybernetics (ICMLC'07), Honk Kong, pp. 3607–3612
37. Nishida K, Yamauchi K (2007) Detecting concept drift using statistical testing, Discovery Science. Springer Berlin Heidelberg
38. Nishida K (2008) Learning and detecting concept drift. PhD Thesis, Hokkaido University, Japan

39. Pinho AG, Vellasco M, Abs da Cruz AV (2009) A new model for credit approval problems: A quantum-inspired neuro-evolutionary algorithm with binary-real representation. Nature & Biologically Inspired Computing (NaBIC). World Congress on. IEEE
40. Pinho AG (2010) Algoritmo evolucionário com inspiração quântica e representação mista aplicado a Neuroevolução. Master's Dissertation, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, (in portuguese)
41. Polikar R, Elwell R (2013) Benchmark Datasets for Evaluating Concept drift/NSE Algorithms. Avaliable at: http://users.rowan.edu/~polikar/research/NSE
42. Ross GJ, Adams NM, Tasoulis DK, Hand DJ (2012) Exponentially weighted moving average charts for detecting concept drift. Pattern Recogn Lett 33(2):191–198
43. Schlimmer J, Granger R (1986) Incremental learning from noisy data. Mach Learn 1(3):317–354
44. Scholz M, Klinkenberg R (2005) An ensemble classifier for drifting concepts. In Proceedings of the 2nd International Workshop on Knowledge Discovery in Data Stream, pp. 53–64
45. Scholz M, Klinkenberg R (2007) Boosting classifiers for drifting concepts. Intelligent Data Analysis 11(1):3–28
46. Sebastião R, Gama J, Mendonça T (2017) Fading histograms in detecting distribution and concept changes. International Journal of Data Science and Analytics:1–30
47. Silveira L, Tanscheit R, Vellasco M (2017) Quantum Inspired Evolutionary Algorithm for Ordering Problems. Expert Syst Appl 67:71–83
48. Stanley KO (2003) Learning concept drift with a committee of decision trees. Department of Computer Sciences, University of Texas at Austin, Tech. Rep. AI-03-302
49. Street WN, Kim YS (2001) A streaming ensemble algorithm (SEA) for largescale classification. In Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 377–382
50. Sun Y, Wang Z, Liu H, Du C, Yuan J (2016) Online Ensemble Using Adaptive Windowing for Data Streams with Concept Drift. International Journal of Distributed Sensor Networks
51. Tsymbal A (2004) The problem of concept drift: Definitions and related work. Tech. Rep
52. Vellasco MBR, Abs da Cruz AV, Pinho AG (2010) Quantum-inspired evolutionary algorithms applied to neural network modeling. In IEEE world congress on computational intelligence (WCCI), pp. 125–150
53. Wozniak M, Kasprzak A, Cal P (2013) Application of combined classifiers to data stream classification. In Proceedings of the 10th International Conference on Flexible Query Answering Systems FQAS 2013, LNCS, page in press, Berlin, Heidelberg, SpringerVerlag
54. I. Zliobaite (2009) Learning under Concept Drift: An Overview. Tech. rep. Vilnius University
55. Jorge PMC (2018) Síntese de Comitê de Árvores de Padrões Fuzzy através da Programação Genética Cartesiana em Ambientes Não EstacionÁrios. MSc Dissertation, State University of Rio de Janeiro, Rio de Janeiro
56. Ferreira RS, Zimbrão G, Alvim LGM (2019) AMANDA: Semi-supervised density-based adaptive model for non-stationary data with extreme verification latency. Inf Sci 488:219–237
57. Krawczyk B, Cano A (2018) Online ensemble learning with abstaining classifiers for drifting and noisy data streams. Appl Soft Comput 68:677–692
58. Ye R, Dai Q (2018) A novel greedy randomized dynamic ensemble selection algorithm. Neural Process Lett 47(2):565–599
59. Cano A, Krawczyk B (2018) Learning classification rules with differential evolution for high-speed data stream mining on GPU s. 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE
60. Cano A, Krawczyk B (2019) Evolving rule-based classifiers with genetic programming on gpus for drifting data streams. Pattern Recogn 87:248–268
61. Angelov PP, Zhou X (2008) Evolving fuzzy-rule-based classifiers from data streams. IEEE Trans Fuzzy Syst 16(6):1462–1475

Tatiana Escovedo received the BSc and MSc degrees in Computer Science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil, in 2005 and 2007, respectively, and the PhD degree in Electrical Engineering from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in 2015. Dr. Escovedo is currently the coordinator of artificial intelligence of Petrobras' digital transformation department in Rio de Janeiro, Brazil, and assistant professor at PUC-Rio. She is the author of several papers in the area of software engineering and machine learning. Her research interests include Data Science, Artificial Intelligence, Software Engineering, Machine Learning and Business Intelligence.

Adriano Soares Koshiyama received his BSc degree in Economics from UFRRJ and MSc degree in Electrical Engineering from PUC-Rio. Nowadays is a PhD Candidate in Computer Science at University College London (UCL), with its main research subject being in Financial Computing and Analytics. Its main research topics are related to: Machine Learning, Statistical Methods, Optimization and Finance.

Andre Vargas Abs da Cruz received the BSc. in Computer Engineering at the Pontifical Catholic University of Rio de Janeiro (1998), MSc. in Electric Engineering (Support Decision Methods) at the Pontifical Catholic University of Rio de Janeiro (2003) and DSc. in Electric Engineering (Support Decision Methods) at the Pontifical Catholic University of Rio de Janeiro (2007). Did a post-doctoral research in bioinformatics. Has experience on the following subjects: optimization, evolutionary algorithms, quantum computing, bioinformatics and neural networks. He currently works as a Data Scientist for MDC Partners in Antwerp, Belgium.

Marley Maria Bernardes Rebuzzi Vellasco received the BSc and MSc degrees in Electrical Engineering from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil, in 1984 and 1987, respectively,and the PhD degree in Computer Science from the University College London (UCL) in 1992. Dr. Vellasco is currently Head of the Electrical Engineering Department of PUC-Rio and of the Computational Intelligence and Robotics Laboratory (LIRA) of PUC-Rio. She is the author of four books and more than 60 papers in professional journals, 340 papers in conference proceedings and 17 book chapters in the area of soft computing and machine learning. Her research interests include Neural Networks, Fuzzy Logic, Neuro-Fuzzy Systems, Neuro-Evolutionary models, Robotics, and Intelligent Agents, applied to decision support systems, pattern classification, time-series forecasting, control, optimization and Data Mining.