International Journal of Computational Intelligence and Applications
Vol. 12, No. 4 (2013) 1340002 (15 pages)

DOI: 10.1142/S1469026813400026

Imperial College Press
www.icpress.co.uk

LEARNING UNDER CONCEPT DRIFT USING A NEURO-EVOLUTIONARY ENSEMBLE

TATIANA ESCOVEDO*, ANDRÉ V. ABS DA CRUZ†,
MARLEY M. B. R. VELLASCO‡ and ADRIANO S. KOSHIYAMA§

Electrical Engineering Department
Pontifical Catholic University of Rio de Janeiro
(PUC-Rio) Rio de Janeiro, Brazil
**tatiana@ele.puc-rio.br*
†*andrev@ele.puc-rio.br*
‡*marley@ele.puc-rio.br*
§*adriano@ele.puc-rio.br*

Received 21 July 2013
Revised 14 September 2013
Published 20 December 2013

This work describes the use of a weighted ensemble of neural network classifiers for adaptive learning. We train the neural networks by means of a quantum-inspired evolutionary algorithm (QIEA). The QIEA is also used to determine the best weights for each classifier belonging to the ensemble when a new block of data arrives. After running several simulations using two different datasets and performing two different analysis of the results, we show that the proposed algorithm, named neuro-evolutionary ensemble (NEVE), was able to learn the data set and to quickly respond to any drifts on the underlying data, indicating that our model can be a good alternative to address concept drift problems. We also compare the results obtained by our model with an existing algorithm, Learn++.NSE, in two different nonstationary scenarios.

Keywords: Adaptive learning; concept drift; neuro-evolutionary ensemble; quantum-inspired evolution.

1. Introduction

The ability for a classifier to learn from incrementally updated data drawn from a nonstationary environment poses a challenge to the field of computational intelligence. Moreover, the use of neural networks as classifiers makes the problem even harder, as neural networks are usually seen as tools that must be retrained with the whole set of instances learned so far when a new chunk of data becomes available.

In order to cope with that sort of problem, a classifier must, ideally, be able to[1]:

- Track and detect any sort of changes on the underlying data distribution;
- Learn with new data without the need to present the whole data set 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 those abilities try, in a way or another, to address a phenomenon called "concept drift".[2,3] This phenomenon defines data sets which suffer changes over time, like, for example, when the relevant variables change or either mean or variance of the time series is changing. Many approaches have been devised in order to accomplish some or all of the abilities mentioned above. The simplest one consists in using a sliding window on incoming data and training the classifier with the data delimited by that window.[4] Another approach consists in detecting drifts and then making adjustments to the classifier according to the drift.

A more successful approach consists in using an ensemble of classifiers. This kind of approach uses a group of different classifiers in order to be able to track changes on the environment. Several different models of ensembles have been proposed in the literature[5–7]:

- Ensembles that create new classifiers to each new chunk of data and weight classifiers according to their accuracy on recent data;
- Unweighted ensembles which can cope with new data that belongs to a concept different from the most recent training data;
- Ensembles that are able to discard classifiers as they become inaccurate or when a concept drift is detected.

Most models using weighted ensembles determine the weights for each classifier using some sort of heuristics related to the amount of mistakes the classifier does when working with the most recent data.[3] Although in principle any classifier can be used to build the ensembles, the ones which are most commonly used are decision trees, neural networks and naive Bayes.[8]

In this work, we present an approach based on neural networks which are trained by means of a quantum-inspired evolutionary algorithm (QIEA). QIEAs[9–13] are a class of estimation of distribution algorithms (EDAs) which present, for several benchmarks, a better performance for combinatorial and numerical optimization when compared to their canonical genetic algorithm counterparts. We also use the QIEA-R to determine the voting weights for each classifier which is part of the ensemble. Every time a new chunk of data arrives, a new classifier is trained on this new data set and all the weights are optimized in order for the ensemble to improve its performance on classifying this new set of data.

Therefore, we present a new approach for adaptive learning, consisting of an ensemble of neural networks, named neuro-evolutionary ensemble (NEVE). To evaluate its performance and accuracy, we used two different datasets to execute several simulations, varying the ensemble settings and analyzing how they influence the final result. We also compare the results of NEVE with the results of Learn++. NSE algorithm,[5] an existing approach to address adaptive learning problems.

This paper is organized in four additional sections. Section 2 presents some theoretical concepts related to concept drift, the quantum-inspired neuro-evolutionary algorithm and the Learn++.NSE algorithm. Section 3 details the proposed model and Sec. 4 presents and discusses the results of the experiments. Finally, Sec. 5 concludes this paper and present some possible future works.

2. Concept Drift Concepts, the Quantum-Inspired Neuro-Evolutionary Model and the Learn++.NSE Algorithm

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. An environment from which this kind of data is obtained is considered a nonstationary environment.

Concept drift can also be defined as an obstacle caused by insufficient, unknown or unobservable features in a dataset, which happens in many real problems. These problems usually depend on a context that is not explicitly stated in the predicted features. This scenario is known as hidden context, and a typical example is the climatic prediction rules, that can vary radically according to the season of the year. Another example would be the analysis of consumption patterns that can vary in time, according to the month, availability of alternative products, inflation rate, etc.[2] Analyzing the problem with the benefit of this hidden context would help solve the nonstationarity problem.

A practical example of concept drift mentioned in Ref. 6 is detecting and filtering out spam e-mails. The description of the two classes "spam" and "nonspam" may vary in time. They are user specific, and user preferences are also varying 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 the "spammers", who will keep creating new forms to trick the classifier into labeling a spam as a legitimate e-mail.

Algorithms designed for concept drift can be characterized in several ways. Table 1, based on Refs. 5–7, summarizes the most commonly used.

The ensemble-based approaches that combine multiple classifiers constitute a new breed of nonstationary learning (NSL) algorithms. These algorithms tend to be more accurate, more flexible and sometimes more efficient than single classifiers.[6] Most of all uses some voting method, yet there is no agreement in the literature about the best type to be used.

Next subsection introduces the quantum-inspired neuro-evolutionary algorithm used to train NEVE.

2.2. *The quantum-inspired neuro-evolutionary model*

Neuro-evolution is a form of machine learning that uses evolutionary algorithms to train artificial neural networks. This kind of model is particularly interesting for reinforcement learning problems, where the availability of input–output pairs is

Table 1. Different types of concept drift algorithms.

Active versus Passive Approach	
Active	Uses some drift detection mechanism, learning only when the drift is detected.
Passive	Assume possibly ongoing drift and continuously update the model with each new data (set). If change has occurred, it is learned, else, the existing knowledge is reinforced.
Online versus Batch Algorithms	
Online	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.
Batch	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 versus Ensemble	
Single classifier	Uses only one classifier.
Ensemble	Combines multiple classifiers.

often difficult or impossible to obtain and the assessment of how good the network performs is made by directly measuring how well it completes a predefined task. As training the weights in a neural network is a nonlinear global optimization problem, it is possible to minimize the error function by means of using an evolutionary algorithm approach.

The QIEA is a class of EDA that has a fast convergence and, usually, provides a better solution, with less evaluations than the traditional genetic algorithms.[6,8] In this model, quantum-inspired genes are represented by probability density functions (PDF) which are used to generate classical individuals through an observation operator. After being observed, the classical individuals are evaluated, as in traditional genetic algorithms, and, by means of using fitness information, a set of quantum-inspired operators are applied to the quantum individuals, in order to update the information they hold in such a way that on the next generations, better individuals will have a better chance to be selected. Further details on how this global optimization method works can be found in Refs. 9–13.

Based on this algorithm, the proposed quantum-inspired neuro-evolutionary model consists in a neural network [a multilayer perceptron (MLP)] and a population of individuals, each of them encoding a different configuration of weights and biases for the neural network. The training process occurs by building one MLP for each classical individual using the genes from this individual as weights and biases. After that, the full training data set (or the set of tasks to be performed) is presented to the MLP and the average error regarding the data set is calculated for each MLP. This average error is used as the fitness for each individual associated to that MLP, which allows the evolutionary algorithm to adjust itself and move on to the next generation, when the whole process will be repeated until a stop condition is reached.

This subsection presented the quantum-inspired neuro-evolutionary model. This model will be the basis for the algorithm presented in Sec. 3.

2.3. *Learn++.NSE*

Developed based on the guidelines for building learning algorithms in nonstationary environments, previously presented in this section, Learn++.NSE[5] is an ensemble-based batch learning algorithm that uses weighted majority voting, where the weights are dynamically updated with respect to the classifiers' time-adjusted errors on current and past environments. The algorithm uses a passive drift detection mechanism, and uses only current data for training. It can handle a variety of nonstationary environments, including sudden concept change, or drift that is slow or fast, gradual or abrupt, cyclical, or even variable rate drift. It is also one of the few algorithms that can handle concept addition (new class) or deletion of an existing class.

The algorithm assumes that at each step may or may not have occurred change in environment and, if occurred, the rate of change is unknown and it is assumed that it is not constant. It is also assumed that all previously seen data (relevant or not for the learning) is not accessible or it is not possible to access it, meaning that the algorithm works incrementally. All relevant information about the previous data must be stored in the parameters of the previously generated classifiers. Depending on the nature of the change, the algorithm retains, builds or temporarily discards knowledge, so that new data can be categorized.

The knowledge base is initialized by creating a single classifier in the first data block available. Once prior knowledge is available, the current ensemble (knowledge base) is evaluated by current data: the algorithm identifies which new samples were not recognized by the existing knowledge base and this is updated by adding a new classifier trained on current training data. Each classifier (including the recently created) is evaluated on the training data. As previously unknown data have been identified, the penalty for misclassifying is considered in the error calculation. This way, more credit is given to the classifiers capable of identify previously unknown instances, while classifiers that misclassify previously known data are penalized. Then, classifier error is weighted considering the time: recent competence is taken more into account when categorizing knowledge. After that, the voting weights are determined: if knowledge of a classifier is not compatible with the current environment, it receives little or no weight and is temporarily removed from the knowledge base. It is not discarded: if its knowledge becomes relevant again, it will receive higher voting weights. The final decision is taken on with the weighted majority vote of the current ensemble members.

3. NEVE: The Neuro-Evolutionary Ensemble

To some applications, such as those that use data streams, the strategy of using simpler models is most appropriate because there may not be time to run and update an ensemble. However, when time is not a major concern, yet the problem requires high accuracy, an ensemble is the natural solution. The greatest potential of this

strategy for detecting drifts is the ability of using different forms of detection and different sources of information to deal with the various types of change.[7]

One of the biggest problems in using a single classifier (a neural network, for example) to address concept drift problems is that when the classifier learns a dataset and then we need it to learn a new one, the classifier must be retrained with all data, or else it will "forget" everything already learned. Otherwise, using the ensemble, there is no need to retrain it again, because it can "retain" the previous knowledge and still learn new data.

Hence, in order to be able to learn as new chunks of data arrive, we implemented an ensemble with neural networks that are trained by an evolutionary algorithm, presented in Sec. 2.2. This approach makes the ensemble useful for online reinforcement learning, for example. The algorithm works as shown in Fig. 1 and each step is described in detail on the next paragraphs.

On step 1 we create the empty ensemble with a predefined size equal to s. When the first chunk of data is received, a neural network is trained by means of the QIEA-R until a stop condition is reached (for example, the number of evolutionary generations or an error threshold). If the number of classifiers in the ensemble is smaller than s, then we simply add this new classifier to the ensemble. This gives the ensemble the ability to learn the new chunk of data without having to parse old data. If the ensemble is already full, we evaluate each classifier on the new data set and we remove the one with the highest error rate (including the new one, which means the new classifier will only become part of the ensemble if its error rate is smaller than the error rate of one of the classifiers already in the ensemble). This gives the ensemble, the ability to forget about data which is not needed anymore.

Finally, we use the QIEA-R to evolve a voting weight for each classifier. Optimizing the weights allows the ensemble to quickly adapt to sudden changes on the data, by giving higher weights to classifiers better adapted to the current concepts

1. Create an empty ensemble P
2. Define the ensemble size s
3. For each chunk of data D_i; $i = 1, 2, 3, \ldots, m$ do
3.1. Train the classifier using the QIEA-R and a MLP and calculate its error E' over the data chunk
3.2. If the ensemble is full (number of classifiers = s) then
   i) Calculate the classification error E_i for each classifier c_i in the ensemble
   ii) If ($E' > max(E_i)$)
      a. Replace the classifier with $max(E_i)$ by the new classifier
3.3. else
   i) Add the new classifier to the ensemble
3.4. Evolve the voting weights w_i for each classifier in the ensemble using the last chunk of data D_i

Fig. 1. The NEVE training algorithm.

governing the data. The chromosome that encodes the weights has one gene for each voting weight, and the population is evolved using the classification error as the fitness function. It is important to notice that when the first $s-1$ data chunks are received, the ensemble size is smaller than its final size and thus, the chromosome size is also smaller. From the s data chunk on, the chromosome size will remain constant and will be equal to s.

In this work we used only binary classifiers but there is no loss of generality and the algorithm can be used with any number of classes. For the binary classifier, we discretize the neural network's output as "1" or "−1" and the voting process for each instance of data is made by summing the NN's output multiplied by its voting weight. In other words, the ensemble's output for one instance k from the ith data chunk is given by

$$P(D_{ik}) = \sum_{j=0}^{s} w_j c_j(D_{ik}), \tag{1}$$

where $P(D_{ik})$ is the ensemble's output for the data instance D_{ik}, w_j is the weight of the jth classifier and $c_j(D_{ik})$ is the output of the jth classifier for that data instance. If $P(D_{ik}) < 0$, we assume the ensemble's output is "−1". If $P(D_{ik}) > 0$, we assume the ensemble's output is "1". *If* $P(D_{ik}) = 0$, we choose a class randomly.

4. Experimental Results

In order to check the ability of our model on learning data sets with concept drifts, we used two different data sets (SEA Concepts and Nebraska, also used in Ref. 5) upon which we performed several simulations in different scenarios.

The SEA Concepts was developed by Street[14] and has been used by several algorithms as a standard test for concept change. The dataset, available in Ref. 15, is characterized by extended periods without any drift with occasional sharp changes in the class boundary, i.e., sudden drift or concept change. The dataset consists of 50000 random points in a three-dimensional feature space. The features are in the [0; 10] domain but only two of the three features are relevant to determine the output class. These points are then divided into four blocks, with different concepts. Class labels are assigned based on the sum of the relevant features, and are differentiated by comparing this sum to a threshold that separates a 2D hyper-plane: an instance is assigned to class 1 if the sum of its (relevant) features (f1 + f2) fall below the threshold, and assigned to class 2, otherwise. At regular intervals, the threshold is changed with increasing severity ($8 \rightarrow 9 \rightarrow 7.5 \rightarrow 9.5$), creating an abrupt shift in the class boundary.

The Nebraska dataset, also available at Ref. 15, presents a compilation of weather measurements from over 9000 weather stations worldwide by the U.S. National Oceanic and Atmospheric Administration since 1930s, providing a wide scope of weather trends. Daily measurements include a variety of features (temperature,

pressure, wind speed, etc.) and indicators for precipitation and other weather-related events. As a meaningful real world dataset, we chose the Offutt Air Force Base in Bellevue, Nebraska, for this experiment due to its extensive range of 50 years (1949–1999) and diverse weather patterns, making it a long-term precipitation classification/prediction drift problem. Class labels are based on the binary indicator(s) provided for each daily reading of rain: 31% positive (rain) and 69% negative (no rain). Each training batch consisted of 30 samples (days), with corresponding test data selected as the subsequent 30 days. Thus, the learner is asked to predict the next 30 days' forecast, which becomes the training data in the next batch. The dataset included 583 consecutive "30-day" time steps covering 50 years.

4.1. *Running details*

All simulations begin at $t = 0$ and end at an arbitrary time $t = 1$. Meanwhile, T consecutive data blocks are presented for training, each one taken from a possible drift scenario, where the rate or nature is unknown. The value T determines the number of time steps (or snapshots) taken from the data during the drift period. A large T corresponds to a low drift rate, while a small T corresponds to a high rate, because the algorithm sees less data snapshots during a given time period.

On each simulation, we used a fixed topology for the neural networks consisting of three inputs for SEA Concepts dataset and eight inputs for Nebraska dataset, representing the input variables for each dataset. In both datasets, we used one output, and we varied the number of the neurons for the hidden layer. Each neuron has a hyperbolic tangent activation function and, as mentioned before, the output is discretized as "−1" or "1" if the output of the neuron is negative or positive, respectively. The evolutionary algorithm trains each neural network for 100 generations. The quantum population has 10 individuals and the classical population 20. The crossover rate is 0:9 (refer to Refs. 9 and 10 for details on the parameters). The same parameters are used for evolving the weights for the classifiers. The neural network weights and biases and the ensemble weights are allowed to vary between −1 and 1 as those values are the ones who have given the best results on some pre-evaluations we have made.

The first experiment was conducted in order to evaluate the influence of the variation of the parameters values in the results (number of the hidden layer neurons and the size of the ensemble). We used four different configurations for each dataset. After running ten simulations for each configuration, we performed some an analysis of variance (ANOVA).[16] In order to use ANOVA, we tested the normality assumption for the noise term with Shapiro–Wilk's test[17] and the homogeneity of variances with Bartlett's test.[16] All the statistical procedures were conducted in R package,[18] admitting a significance level of 5%.

The second experiment, in turn, aimed to compare the results found by NEVE and Learn++.NSE algorithms and we used, for each dataset, the best configuration found by first experiment (ensemble size and number of neurons at hidden layer

values). After running one simulation for each dataset, we made statistical comparisons between the results found by NEVE and Learn++.NSE algorithms. The results of Learn++.NSE can be found at Ref. 5. Then, to evaluate NEVE we made 10 runs for each dataset used, due to the stochastic optimization algorithm used to train NEVE. Based on these runs, we calculate some statistical parameters (mean, standard deviation, etc.) that were used to compute the Welch t-test[19] to evaluate which algorithm had, in average, the best performance in test phase. The normality assumption necessary for Welch t-test was verified using Shapiro–Wilk test.[17] All the statistical analyses were conducted in R statistical package.[18]

4.2. *First experiment*

Based on the past subsections, we made 40 simulations using SEA dataset and 40 simulations using Nebraska dataset, using 4 different configurations on each dataset. Table 2 displays number of neurons at hidden layer and ensemble size with different levels (5 and 10) and the output (average error in test phase for 10 runs) for each configuration.

In order to evaluate which configuration provided a significant lower error, we have to perform multiple comparisons between the results of each configuration. For each dataset if we decide to use t-test[16] for example, we have to realize six comparisons between the configurations, and thus, the probability that all analysis will be simultaneous correct is substantially affected. In this way, to perform a simultaneous comparison between all configurations we fitted a one-way ANOVA[16] for each dataset, described by

$$Y_{ij} = \mu + \mathrm{CF}_j + \varepsilon_{ij};\ \varepsilon_{ij} \sim N(0, \sigma^2), \tag{2}$$

where Y_{ij} is the ith output for the jth configuration, μ is the global mean, CF_j is the run configuration with j-levels ($j = 1, 2, 3, 4$) for each dataset (A, B, C and D, and E, F, G, H, for SEA and Nebraska, respectively) and ε_{ij} is the noise term, normal distributed with mean zero and constant variance ($\sigma 2$). If CF_j is statistically

Table 2. Results for SEA and Nebraska dataset.

			Error in Test Phase	
Neurons Number	Ensemble Size	Config	Mean	Std. Dev.
SEA dataset				
10	10	A	24.99%	0.17%
5	5	B	24.88%	0.19%
10	5	C	25.06%	0.21%
5	10	D	24.75%	0.17%
Nebraska dataset				
10	10	E	32.30%	0.48%
5	5	F	32.85%	0.43%
10	5	G	33.04%	0.37%
5	10	H	32.10%	0.46%

Table 3. Results for SEA and Nebraska dataset.

Method	Test Statistic	p-Value
ANOVA — SEA dataset		
Bartlett's test	0.4443	0.9309
Config	5.4360	0.0035
Shapiro–Wilk's test	0.9260	0.2137
ANOVA — Nebraska dataset		
Bartlett's test	0.6537	0.8840
Config	11.5900	< 0.0001
Shapiro–Wilk's test	0.9708	0.3803

significant, then some configuration demonstrated an average error different from the others. To verify which configuration has the average error less than other configuration we used Tukey's test.[16]

Then, we analyzed both datasets and exhibit the main results in Table 3.

Analyzing the results displayed in Table 3, in both datasets the errors variance is homogeneous (Bartlett's test, p-value > 0.05). After verifying these two assumptions (normal distribution and homogeneity of variances), we fitted the one-way ANOVA. In both datasets, some configurations demonstrated an average error different from the others (p-value < 0.05). In addition, in both fitted models, the noise term follows a normal distribution (Shapiro–Wilk's test, p-value > 0.05).

In order to identify which configuration performed, in average, better than other, we made Tukey's test for difference of means. Table 4 present the results of this analysis.

It seems that in SEA dataset the D configuration performed significantly better than A and C, although its results is not statistically different than configuration B. In Nebraska the E and H configurations obtained error measures substantially lower than F and G. In fact, we can choose the configuration B as the best configuration to SEA and E to Nebraska dataset, considering the fixed parameters displayed in Table 2. This choice is based on two criterias: lower average error and computational cost to train these models.

Table 4. Tukey's test for SEA and Nebraska dataset.

SEA Dataset			Nebraska Dataset		
Config	Mean Difference	p-Value	Config	Mean Difference	p-Value
A-B	0.11%	0.5665	E-F	−0.55%	0.0145
A-C	−0.07%	0.8018	E-G	−0.74%	0.0013
A-D	0.24%	0.0317	E-H	0.20%	0.8849
B-C	−0.18%	0.1399	F-G	−0.19%	0.7434
B-D	0.13%	0.4010	F-H	0.75%	0.0014
C-D	0.31%	0.0029	G-H	0.94%	0.0001

4.3. *Second experiment: NEVE x Learn++.NSE*

4.3.1. *SEA concepts: NEVE x Learn++.NSE*

Aiming to enable a better comparison with the results of the algorithm Learn++.NSE,[5] detailed in Sec. 2, we used 200 blocks of size 25 to evaluate the algorithm in the test phase. The best configuration previously achieved was five neurons in hidden layer and the size of the ensemble equal to 5. Then, NEVE and Learn++.NSE results were displayed in Table 5.

As can be seen, the mean accuracy rate of NEVE is greater than the best configuration of Learn++.NSE, and thus this difference is statistically significant (tcrit = 18.26, p-value < 0.0001), demonstrating that NEVE performed better, in average, than Learn++.NSE. Figure 2 presents results obtained in the Learn++.NSE algorithm, comparing with other existing algorithms in the literature. Further details can be found in Ref. 5. Figure 3 presents results obtained in the NEVE algorithm.

Next subsection exhibits a second experiment, based on Nebraska Weather data.

4.3.2. *Nebraska weather prediction data*

With the goal to enable a comparison with the results of the algorithm Learn++.NSE, we performed similarly to that used in Ref. 5. The best configuration previously achieved was 10 neurons in hidden layer and the size of the ensemble equal to 10. Then, NEVE and Learn++.NSE results were displayed in Table 6.

As can be seen, the mean accuracy rate of Learn++.NSE is greater than the best configuration of NEVE, and thus this difference is statistically significant

Table 5. Results of the SEA experiments.

Algorithm	Mean	Standard Deviation
NEVE	98.21%	0.16%
Learn++.NSE (SVM)	96.80%	0.20%

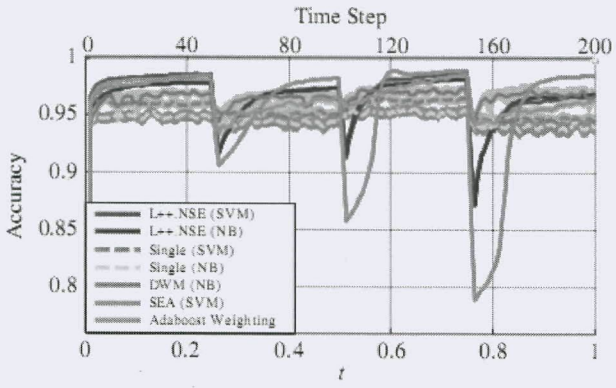

Fig. 2. Comparing the performances of different algorithms using the SEA dataset.[5]

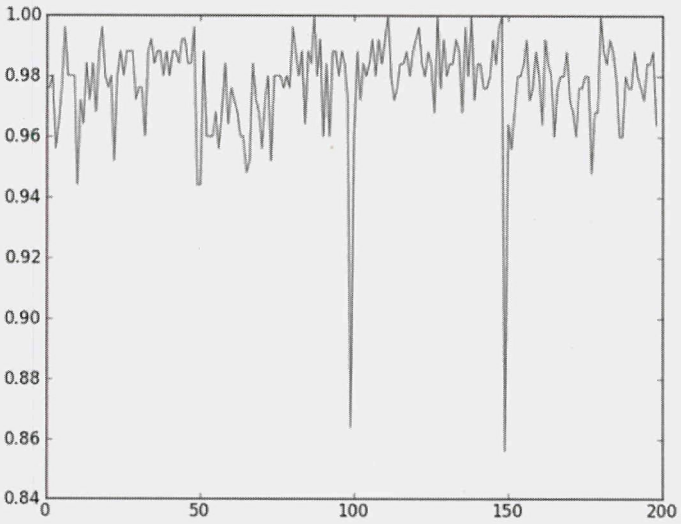

Fig. 3. Evolution of NEVE hit rate in SEA testing set.

Table 6. Results of the NEBRASKA experiments.

Algorithm	Mean	Standard Deviation
NEVE	68.57%	0.46%
Learn++.NSE (SVM)	78.80%	1.00%

(tcrit $= -41.07$, p-value < 0.0001), demonstrating that Learn++.NSE performed better in the test phase on this dataset. Figure 4 present results obtained in the Learn++.NSE algorithm, comparing with other existing algorithms in the literature.

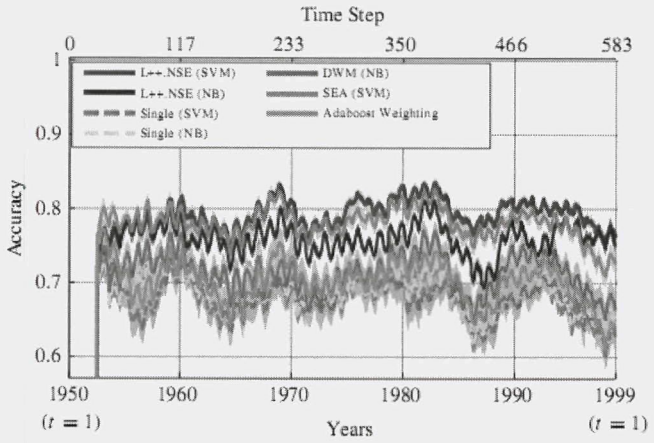

Fig. 4. Comparing the performances of different algorithms using the Nebraska weather prediction dataset (see Ref. 5).

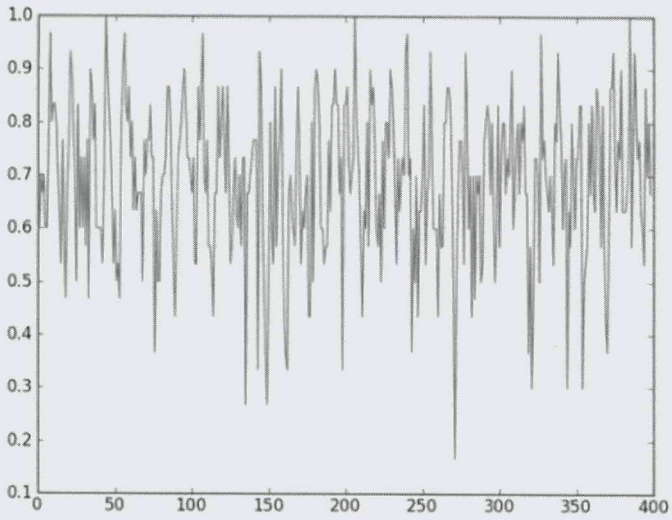

Fig. 5. Evolution of NEVE hit rate in Nebraska testing set.

Further details can be found in Ref. 5. Figure 5 illustrates similarly, but with results of NEVE.

This section presented the results obtained in experiments taken with the NEVE algorithm. We detailed the results of two different datasets and then those results were compared with the results of the algorithm Learn++.NSE. The next section concludes this work.

5. Conclusions and Future Works

This paper presented a model that uses an ensemble of neural networks trained by a QIEA to learn datasets (possibly with concept drifts) incrementally.

We analyzed the ability of the model using two different datasets and conducting two different experiments. In the first experiment, we found a good configuration for both datasets and demonstrated how the number of neurons and the ensemble size affect the average error produced by the model. As stated in the results, the ensemble size affected almost two times more the results of NEVE than the number of neurons. In the second experiment, the NEVE algorithm have demonstrated a better performance in SEA dataset compared to Learn++.NSE and yet lower accuracy when comparing with Learn++.NSE in Nebraska dataset.

Although the NEVE algorithm have demonstrated satisfactory performance for the datasets used in the analysis of this study, it is strongly recommended to perform further tests — using different configurations, different datasets and performing different analysis — to confirm the results presented here. We still need to investigate other factors related to QIEA-R fine tuning (genetic operators, population size, etc.).

Checking the performance of NEVE on other real data sets is also a possibility for a future work, although it is not easy to determine if a real world data set has any kind of significant changes on data. In any case, it is always possible to introduce these changes on any real data set, artificially. We also intend in the future to continue this work, analyzing other existing approaches, such as Ref. 20 and 21, and performing new experiments in comparison with these and other algorithms.

References

1. J. C. Schlimmer and R. H. Granger, Incremental learning from noisy data, *Mach. Learn.* **1** (1986) 317–354.
2. A. Tsymbal, The problem of concept drift: Definitions and related work, *Tech. Rep.* (2004).
3. M. T. Karnick, M. Ahiskali, M. Muhlbaier and R. Polikar, Learning concept drift in nonstationary environments using an ensemble of classifiers based approach, *Int. Joint Conf. Neural Networks* (Hong Kong, China, 2008), pp. 3455–3462.
4. G. Hulten, L. Spencer and P. Domingos, Mining time-changing data streams, *ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* (New York, NY, USA, 2001), pp. 97–106.
5. R. Elwell and R. Polikar, Incremental learning of concept drift in nonstationary environments, *IEEE Trans. Neural Netw.* **10** (2011) 1517–1531.
6. L. I. Kuncheva, *Multiple Classifier Systems*, eds. F. Roli, J. Kittler and T. Windeatt (Springer, New York, 2004), p. 383.
7. L. I. Kuncheva, Classifier ensemble for detecting concept change in streaming data: Overview and perspectives, in *Proc. Eur. Conf. Artif. Intell.*, Vol. 2008 (2008), pp. 5–10.
8. N. C. Oza, Online ensemble learning, Dissertation, University of California, Berkeley (2001).
9. A. V. Abs da Cruz, M. M. B. R. Vellasco and M. A. C. Pacheco, Quantum-inspired evolutionary algorithms for numerical optimization problems, in *Proc. IEEE World Conf. Computational Intelligence* (Vancouver, BC, Canada, 2006).
10. A. V. Abs da Cruz, Quantum-inspired evolutionary algorithms for optimization with numeric representation, Ph.D. dissertation, Pontifical Catholic University, Rio de Janeiro (2007) (in Portuguese).
11. K. H. Han and J. H. Kim, Quantum-inspired evolutionary algorithm for a class of combinatorial optimization, *IEEE Trans. Evol. Comput.* **6** (2002) 580–593.
12. K. H. Han and J. H. Kim, On setting the parameters of QEA for practical applications: Some guidelines based on empirical evidence, *Genetic and Evloutionary Computation Conf.* (Chicago, Illinois, USA, 2003), pp. 427–428.
13. K. H. Han and J. H. Kim, Quantum-inspired evolutionary algorithms with a new termination criterion, He gate, and two-phase scheme, *IEEE Trans. Evol. Comput.* **8** (2004) 156–169.
14. W. N. Street and Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in *Proc. 7th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining* (San Francisco, CA, USA, 2001), pp. 377–382.
15. R. Polikar and R. Elwell. Benchmark datasets for evaluating concept drift/NSE algorithms, http://users.rowan.edu/~polikar/research/NSE.
16. D. C. Montgomery, *Design and Analysis of Experiments* (Wiley, New York, 2008).
17. P. Royston, An extension of Shapiro and Wilk's W test for normality to large samples, *Appl. Stat.* **31** (1982) 115–124.

18. R Development Core Team, *R: A Language and Environment for Statistical Computing* (R Foundation for Statistical Computing, Vienna, 2012), www.r-project.org.
19. P. Dalgaard, *Introductory Statistics with R* (Springer-Verlag, New York, 2002).
20. J. Kolter and M. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts, *J. Mach. Learn. Res.* **8** (2007) 2755–2790.
21. K. Jackowski, Fixed-size ensemble classifier system evolutionarily adapted to a recurring context with an unlimited pool of classifiers, *Pattern Anal. Appl.* **8** (2013) 1–16.