Assessing the classification accuracy of an evolutionary neural network: the case of plant virus identification

Thomas J. Glezakos, Georgia Moschopoulou, Theodore A. Tsiligiridis, Spiridon Kintzios, and Constantine P. Yialouris

***Abstract*— Genetic algorithms and neural networks are exciting technologies in the fields of artificial intelligence, machine learning and data mining. In this work, we propose an evolutionary method to produce time series meta-data for the training and testing of a neural network module used in plant virus identification. The initial dataset is derived via a prototype method developed by members of our research team, which uses specially designed sensors to monitor the virus reactions. The proposed method incorporates an evolutionary technique to manipulate the time series in order to effectively manipulate the dimensionality of the input data space. Furthermore, our method is tested against some of the most commonly used classifiers in machine learning and proves its potential towards assisting an identification which, up to now, is accomplished mainly empirically.**

I. Introduction

THE analysis and feature extraction of time series data most often than not, resorts to regression models, such as the autoregressive and the moving average methods, which are inhibited by the non-linearity inherent in the input data space. Numerous fault-tolerant tools, such as fuzzy systems and neural networks have been engaged in order to address this problem, as in [6], [13]-[14], [16]-[17], [20]-[21], [29], [32], [34], [36]–[39]. Also, a lot of research has been dedicated in studying the development and prototyping of neural network design or the training and testing via meta – heuristic methods, such as in [1], [12], [30], [31], [33], [35]. Finally, the multi – classification systems design and prototyping has been contemplated in the recent years, in order to provide models which might be rendered as potential problem solvers. In this latter case, multiple classification techniques and models are combined in individual systems trained to provide solutions to pattern recognition problems. Classifier combination approaches might be categorized mainly over three dimensions in this context: the representational and the architectural methodology, as well as the learning technique [2], [27]. All these combinatorial methods might potentially provide models able to discern through the input space and offer feasible solutions to problems which, either cannot be solved by single classifier powered systems, or which might be more effectively handled by a multi classifier one.

The dimensionality of the input data vector has been contemplated a lot in this context and has been found to be most crucial for this kind of data analysis. Most of the researchers agree that in case the dimensionality may be delivered too small, the neural network might not have all the important information at its disposal. On the other hand, if the dimensionality rises at high levels, then we run the risk of over - feeding the network, which might result in redundant information and noise creeping into it, inhibiting its function, causing over - fitting and smothering its ability to generalize [28].

Our research in its essence describes the design, implementation and testing of an innovative method to overcome the problems that the dimensionality of the input data vector might pose on the classification power of neural network systems dealing with time series. It also assesses the classification potential of the evolutionary neural system produced, towards several widely used classifiers, trained and tested with the same data set, by setting a contest between the former and the latter. The system introduced in our work is an evolutionary trained Multi Layer Perceptron (MLP) used in the line of detecting plant viruses using time series data produced by bio-sensors.

The key feature of our system is the prototyping of an evolutionary method to produce genetically enhanced meta – data from the initial time series data set, which may produce a more powerful trained network. Thus, the research initializes by several testing techniques to design an MLP able to use the initial input space as a training pool and produce a rough neural classifier but, instead of training and testing the produced MLP with the whole initial data set, we alternatively engage an evolutionary method to genetically manipulate the time series. In this way, we come up with the production of meta – data for training and testing, which introduce vast enhancements to our neural system as regards both to its discerning capabilities, as well as to its generalization potential. The system thus produced enters the contest arena to compare to other widely used systems, proving its classification potential towards classifiers like the naïve Bayesian, the k Nearest Neighbors and the classification trees, all of which attack the same complex problem of plant virus identification.

All authors are with the Agricultural University of Athens, Iera Odos 75, 118 55, Athens, Hellas

T. Glezakos, T. Tsiligiridis and C.P. Yialouris are with the Scientific Department of the University, Laboratory of Informatics (phones: +30 210 5294181, 176, 182 e-mail: t_glezakos@yahoo.com, tsili@aua.gr, yialouris@aua.gr respectively)

G. Moschopoulou and S. Kintzios are with the Agricultural Biotechnology Department, Laboratory of Plant Physiology and Morphology (e-mail: skin@aua.gr, geo_mos@aua.gr respectively)

II. MATERIALS AND METHODS

A. *Bioelectric Recognition Assay (BERA).*

The Bioelectric Recognition Assay (BERA) is a novel method using biosensors to identify various chemical and molecular structures. The method essentially assesses the structures' interactions with a group of cell components immobilized in a gel matrix preserving their physiological functions. The structures to be identified are referred to as 'ligands' in the sense that they are usually smaller molecular units – i.e. viruses – which specifically bind to the larger biosensor cells. This procedure ultimately results in the alteration of the reagent physiology and the emittance of electrical energy. Recent studies [22-26] have revealed the usability of the method for cheap and fast identification of human infectious viruses and its potential to replace more time - consuming and costly methods, such as the reverse transcription polymerase chain reaction (PT – PCR).

The method is also massively propelled forward due to the fact that the technology behind the biosensor production is advancing by major leaps. After having gone through a number of improving biosensor generations, BERA sensors were radically redesigned to reach the fifth generation of their design development, which incorporates sensors almost ideal for diagnostic applications. In this fifth generation we shall meet sensors of extremely reduced size, consisting of a disposable array of gel beds loaded with reagent cells. Their production is characterized by cost at the lowest levels and a very high rate of reproducibility and speed of manufacturing. Moreover, the fifth sensor generation has achieved to reduce assay time to approximately twelve seconds. The BERA biosensor method has already been successfully implemented to numerous applications, mainly related to both human and plant virus identification, while its evident potential has been recorded into various bibliographical references.

B. *Data Acquisition and Analysis*

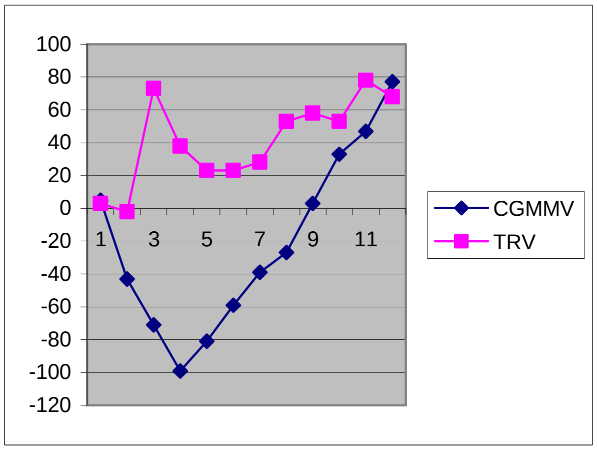

Figure 1. Time series data produced by the BERA sensors for the two viruses investigated.

In this work, the BERA method has been engaged so as to produce data sets which will be used in the detection and identification of certain plant viruses, namely the tobacco rattle virus (TRV) and the cucumber green mottle mosaic (CGMMV) one, using appropriately preprocessed reagents as the sensing elements. While reacting to the biosensors, each of the viruses in question exhibit unique patterns of biosensor responses over specific ranges of concentrations, rendering these responses as a special characteristic for each virus, a real identification signature. Each signature is in essence a graphical curve of bioelectrical responses in the time unit, a time series data set, which should be identified as a characteristic for each virus.

Fig. 1 depicts a close - up of two such signatures / time series, one for each virus, with a resample rate of 6 time units. The x-axis represents the time, while the y-axis' units are voltages produced by the reaction of the biosensor reagent towards the presence of each virus. Up to now these 'signatures' are assessed by experts, special biologists who are able to discern one signature from the other and decide on the identity for each virus. Our effort is focused on producing an expert system which will be trained with various patterns of signatures and their already known outcome, and generalize on any unknown input data space with as higher classification capability as possible.

The data set gathered by the biosensors consisted of 1274 instances of time series signatures for both the tobacco rattle virus and the cucumber green mottle mosaic virus, while each time series consisted of 331 time fragments. That is, for each virus, each biosensor was engaged to produce results for 331 seconds and these were used in order to form the 'signature' of the identifiable object. The constructed data set was used as an instructional scheme for a special designed evolutionary neural network classifier, employing the ten-cross-validation training and testing technique, with a proportion of 70/30.

C. *The Python Programming Language enhanced with the Fast Artificial Neural Network (FANN) and the Orange Libraries.*

Our system is in essence a genetically trained neural network with the architecture of a Multilayered Perceptron. It was developed using the programming language of Python, enhanced by the freely available open source Fast Artificial Neural Network Library, which was especially wrapped up for use with the language. The system was built up from scratch and is in an open source state, meaning it is freely and openly available to everyone. Python is a programming language with a relaxed learning curve, while remaining powerful and incorporating efficient high level data structures as well as a simple and effective approach to object oriented programming. Being a multi platform language, it allows for programs to be developed on most of the platforms available today. Our source code was writen in Ubuntu Linux, but can be run on a windows – based system just as easily. It is also supported by a vast number of freely available libraries, one of which is the Fast Artificial Neural Network Library (FANN), with support for both fully and sparsely connected networks. Cross-platform execution in both fixed and floating point types are supported, while it also includes a framework for easy handling of training data sets. On the other hand, the rest of the classifiers with which our system was compared were constructed via the Orange library [19], a component - based data mining software including a range of preprocessing, modelling and data exploration techniques. Via the orange library the construction of classifiers such as the naïve Bayesian, the decision tree and the k Nearest Neighbor ones, were made possible.

Our system has been successfully implemented in [15] and is used here without alteration. The evolutionary process has been designed in Python from scratch and incorporates a genetic algorithm in order to produce fitter meta – data out of the initial raw time series ones. The genetic algorithm assesses the potential of each designed neural network, engaging its fitness function for that matter.

D. *Artificial Neural Networks*

The human brain, is a highly complicated biologic machine, capable of solving innumerable kinds of problems, from the most simplistic to highly complex ones. Through years of research and experimentation we have come to a state where we can decipher some of its basic operations, though we are yet far from fully understand its mechanisms. The brain is part of the central nervous system, consisting of a large number of interconnected simplistic processing elements called neurons. Each neuron's core, the nucleus, communicates with other neurons by means of the dendrites and the axon, connection which is called a synaptic connection. The neuron fires electric pulses through its synaptic connections which are received by the connections of other neurons at the dendrites level. When a neuron receives enough signalling, that is when the total signals overcome a certain threshold, the neurons fires in turn a similar signal to its counterparts. This way information propagates through the network. The connections, as well as the threshold change throughout the lifetime of each neuron and this is the main reason that the network ultimately learns.

The human brain consists of around 10^{11} neurons which communicate with around 10^{15} connections [18], activating in parallel reacting to various external and internal sources. The brain receives information from the five senses and controls the muscles. Artificial Neural Networks (ANNs) were developed in an attempt to simulate the function of the human brain. An ANN is a software device consisting of a number of simple processing elements interconnected and operating in parallel. Each neuron is only aware of the signals it receives from other connected neurons and the information it sends from time to time to other processing elements. In this context, an ANN is a computer program capable of learning from examples through iteration. In most of the times no prior knowledge of the input data is required, because the training process is essentially a search for the best synaptic weight vector. Learning is the process of adapting or modifying the neurons' connection weights in response to stimuli presented as inputs requiring the presence of a known output. This process enables the network to learn to solve problems by adequately adjusting the strength of the connections between their processing elements according to the input data and the desired outputs. ANNs have been vastly used for recognizing patterns in the input data space or to extract simple rules for complex problems according to their inputs. The key factor in neural network training is generalization, i.e. the capability of the network to predict "unseen" inputs merely with the knowledge that has been acquired during the training process, in which a stimulus presented in the output corresponds to a desired response for a given input. Although a simple processing element (a neuron) may have very limited learning capabilities, there is a spectacular upgrade in computing power when a large number of thus constructed simple processing elements interconnect and collaborate in parallel. In its typical structure, a neural network consists of the input layer the hidden layer and the output layer. The input layer consists of neurons equal to the problem inputs, whereas the hidden layer is the real processing machine of the network, which may be divided in one or more sub-layers. Finally, the output layer produces the outcome of the network.

The ANN concept is currently widely used in various research works, ranging from pattern recognition, quality control, classification and have gained wide recognition in modelling many processes in engineering, such as in [4]-[9]. Lately they have been used to predict wood water isotherm or sorption isotherms in food science [10], [11] and numerous other disciplines. A neural network may learn by example and outmatches rivalling techniques in that it may use its knowledge under untrained circumstances incorporating a large number of variables [18].

E. *Genetic Algorithms*

Genetic algorithms are inspired by evolutionary biology, and especially driven by the Darwinian axiom of the "survival of the fittest", incorporating numerous biological procedures such as inheritance, selection, crossover (otherwise referred to as recombination) and mutation. They are mostly implemented as computer simulations which search the input plane for 'better' solutions to a given optimization problem. The genetic algorithm starts out with an, often random, initial population of encoded representations of candidate solutions. These representations are referred to as chromosomes, genotypes or genome, while the candidate solutions are referred to as phenotypes. The algorithm proceeds by generations each of which is comprised by genotypes of the previous generation, which are elected basically by their fitness and modified on the grounds of a possible recombination or mutation to form a new population of higher overall fitness.

III. THE CONTESTANTS

A. *Evolutionary Multilayer Perceptron*

The innovative classifier that has been developed and has been put to test against all others follows the multilayer perceptron architecture with fully connected layers. The code we developed was enhanced with a neural network object created from the FANN Python library, which permits for a vast number of parameterization on the neural network that it handles. The neural object developed consists of an input layer of varying number of units, conforming to the meta-data produced by the evolutionary procedure and which form the data set used in the supervised learning of the neural classifier. The variability of the input layer of our neural object was dictated by the genetic algorithm producing the meta-training data set. According to this, each trainer produces different input data, in an effort to handle the dimensionality of the input space. Thus, it was unavoidable for the research team to create a 'plastic' input layer, able to adapt to the length of the input vector for each generation.

The hidden layer of our MLP network consists of 25 hidden units, bearing the Sigmoid Symmetric activation function. Table 1 depicts the activation functions tested during the design of the structure of the neural network, with the Sigmoid Symmetric function giving the best results.

Table 1. Activation Functions used

Activation Function	Dependent Variable Span	Description
Linear	-inf < y < inf	y = x*s, d = 1*s
Threshold		x < 0 -> y = 0, x >= 0 -> y = 1

Sigmoid	0 < y < 1	y = 1/(1 + exp(-2*s*x))
		d = 2*s*y*(1 - y)

Sigmoid Symmetric (Tanh)	-1 < y < 1	y = tanh(s*x) = 2/(1 + exp(-2*s*x)) – 1
		d = s*(1-(y*y))

Gaussian	0 < y < 1	y = exp(-x*s*x*s)
		d = -2*x*s*y*s

Gaussian Symmetric	- 1 < y < 1	y = exp(-x*s*x*s)*2-1
		d = -2*x*s*(y+1)*s

Where x is the input to the activation function, y is the output, s is its steepness and d is the derivation (http://leenissen.dk/fann).

The output layer of the neural object consists of two units so as to classify the input space to one of the two viruses as an output.

The training of the network was dictated by the Rprop training algorithm which outperformed in our tests both the Incremental Training and the Simple Batch Training algorithms. The Rprop algorithm is a branch of the batch training ones which update the weight table once after the whole epoch has been completed, in contrast to the incremental algorithm which updates the weights immediately after each pattern has been shown to the network.

In its first phase, our system uses the Root Mean Square (RMS) error of the neural object to form the fitness function of the genetic algorithm governing the production of meta-data. Having initially set an ideal RMS error to be reached, the distance of the recorder error of each trainer from it should suffice to stand as the fitness function deciding the potential of the chromosomes of the evolutionary process. In the next and final phase, the neural thus produced is tested using 10-fold cross validation on testing data conforming to the structure of the winning chromosome of the evolutionary process.

Notes on the structure of the Neural Object. As already mentioned, our neural object utilized 25 neurons in its hidden layer. It was clear by the vast number of the input record set (time series of 331 time stamps for each pattern) that we needed a neural network strong enough to cope with such a high dimensionality. The decision about the optimal structure of the neural network object was made using an automatic search procedure of the FANN library, known as *cascade training on data*, which determines the optimal number of hidden layers and units based on sequential training. By using the initial raw training and testing record set, while leaving the input and output layers intact, we instructed the network to start out its training with one hidden layer containing only one neuron. After having reached an arbitrary set number of iterations – we used 1,000 for our purpose – the system recorded its root mean square error for the testing record set. Then, it added one more neuron to the hidden layer and recorded the RMS error anew, continuing adding neurons one by one to the hidden layer, until a maximum number of neurons was reached. For our research we set 35 maximum neurons for each hidden layer. The neural object thus developed, has the ability to cascade - train on data, a procedure which permits automatic search for the optimal number of hidden units. Thus, the network starts out its training empty in the hidden layer and then, as training continues, it adds neurons one by one and layer by layer, until an optimal neural network structure is reached (http://www.leenissen.dk/fann). Our tests showed that the best structure for the neural object was the one bearing one hidden layer of 25 neurons.

B. Naive Bayesian Classifier

The Bayes' Theorem is entangled in probability theory as a rule by which conditional probabilities are related. Let A and B denote two stochastic events. Then P(A) and P(B) are the marginal (or else prior) probabilities that they will occur independently. These probabilities do not take into account one another. Also, P(A|B) gives the conditional probability that A occurs given B. The Bayes' theorem sets a relation between P(A|B) and P(B|A), as follows:

$$P(A\,|\,B) = \frac{P(B\,|\,A)P(A)}{P(B)}$$

The theorem is valid in all the interpretations of probability and also helps form rules of how synthesize or update evidence-based beliefs in light of new, a posteriori evidence.

A naïve Bayesian classifier is a derivative assumption of the aforementioned theorem, with strong independence assumptions. Such classifiers may be trained very efficiently in supervised learning schemes and further be applied successfully to real world problems. A great advantage of these classifiers is that, for most of the problems, but not independent from that, they require relatively smaller training data sets. The model of any classifier relates C, a dependent variable with a known number of outcomes (or classes), with several feature variables Fi, i = 1, .., n, such as:P(C|F1, …, Fn). Using the Bayes theorem we can write:

$$F(C \mid F_1,...,F_n) = \frac{P(C)P(F_1,...,F_n \mid C)}{P(F_1,...,F_n)}$$

We normally are interested only in the nominator, since the denominator is considered constant, not depending on C. So:

$$P(C \mid F_1,...,F_n) = P(C)P(F_1,...,F_n \mid C) =$$
$$= P(C)P(F_1 \mid C)P(F_2,...,F_n \mid C,F_1) =$$
$$= P(C)P(F_1 \mid C)P(F_2 \mid C,F_1)P(F_3,...,F_n \mid C,F_1,F_2)$$

and so forth.

Assuming that each feature Fi is conditionally independent of every other feature Fj for i≠j, that is assuming naïve conditional independence for the features Fi, we can write:

$$P(F_i \mid C,F_j) = P(F_i \mid C)$$

thus, the above model could be expressed as:

$$P(C \mid F_1,...,F_n) = P(C)P(F_1 \mid C)P(F_2 \mid C)...P(F_n \mid C) =$$
$$= P(C)\prod_{i=1}^{n} P(F_i \mid C)$$

It is obvious that the conditional distribution of class variable C, under the above naïve assumptions, can be expressed as:

$$P(C \mid F_1,...,F_n) = \frac{1}{Z} P(C)\prod_{i=1}^{n} P(F_i \mid C)$$, where Z is a scaling factor dependent only on the values of the features *Fi* and constant if the latter are known.

The naïve Bayes classifier combines the above model with decision rules, the most common of which is the Maximum a Posteriori (MAP) decision rule. The resultant classifier is given by the following function

$$classify(f_1,..., f_n) =$$
$$= \arg\max_c P(C = c)\prod_{i=1}^{n} P(F_i = f_i \mid C = c)$$

C. Classification Trees

Classification Trees, also referred to as Decision Trees or Regression Trees, are also predictive models, which engage a mapping of the features affecting the status of a class variable to conclusions about its target value. They essentially are generators of rules referring to the condition of the class they predict, which are clear and can be easily understood and explained. The tree grows up due to a technique which is called 'binary recursive partitioning'. According to this procedure, the input data is iteratively split into partitions of clear meaning, then each partition is further split into new ones and so on, until a class value is finally met. A tree thus built has leaves for class values and branches for conjunctions of features which lead to these values.

Classification trees are models which require supervised learning in order to learn from example and be able to generalize on real world problems. A trained classification tree learner is transformed into a classifier for a given problem. Thus, given a training set, the classification value is known for each record. The algorithm systematically breaks up the initial recordset into a number of partitions such that the diversity of the class value is minimal within each partition. This procedure produces homogeneous partitions as regards to the feature class in question. The process is repeated for all the fields of the training set and continued at each next node, until a full tree has been built.

A trained classification tree is evaluated via a testing set of patterns. Pruning of leaves and branches has been introduced, as a technique to help the tree improve on classification on the test and the ensuing real world application phases, reducing the over-fitting of the classifier. During the training phase, the algorithm adds a vast number of branches and leaves onto the tree, as a result of the partitioning of the input data plane. It has been noted that most of the times, a non relevant number of such nodes is embedded due to the need of the tree to explain all of the training data set and, as a result, renders the tree prone to data over-fitting. Pruning of these irrelevant nodes helps the tree perform better on real world problems, helping enhance the generalization capabilities of the classifier.

D. k-Nearest Neighbor Classifier

The k-Nearest Neighbor (k-NN) classifier is included among the simplest in apprehension and the hardest in implementation algorithms in machine learning problems. The algorithm is based on the principle of proximity resemblance, i.e. nearest objects are more possible to be alike than farther situated ones. Thus, an object in the input plane is classified according to the majority vote of its k nearest neighbors. The parameter k, standing for 'kriging', is the most vital in the implementation of the algorithm, deciding the crucial distance in which proximity resemblance takes effect. Its optimal estimation is very hard to come by and is made feasible only through try and error techniques or cross – validation. Generally, larger values of k reduce the possibility of noise creeping in classification, but render the boundaries among the values of the class to be predicted vaguer.

The kriging approach has resulted from several geostatistical techniques attempting to interpolate the value of a feature of a certain location given the values of the same features of its nearby locations. Let *Z* be the feature we study and x_i various locations of the input space for which we know the values $z_i = Z(x_i)$, $i = 1, \ldots, n$. We need to predict the value of Z at an unobserved location x_0 , that is we have to find $z_0 = Z(x_0)$. The method computes the issue value based on a stochastic model of the spatial dependence estimated either by a variogram or by the mean value $\mu(x) = E[Z(x)]$ and the covariance $c(x,y)$ of the location in question. The kriging estimator is given by:

$$\widehat{Z}(x_0) = \sum_{i=1}^{n} w_i(x_0) Z(x_i)$$

which is in fact a linear combination of $z_i = Z(x_i)$, i.e. the feature observed values in various locations, normalized by the weight vector $w_i(x_0)$ $i = 1, \ldots, n$. The weight vector is chosen such that the kriging error

$$\sigma_k^2(x_0) = Var(\widehat{Z}(x_0) - Z(x)) =$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} w_i(x_0) w_j(x_0) c(x_i, x_j) +$$
$$+ Var(Z(x)) - 2\sum_{i=1}^{n} w_i(x_0) c(x_i, x_0)$$

is minimized in accordance to the following condition:

$$E[\widehat{Z}(x) - Z(x)] = \sum_{i=1}^{n} w_i(x_0)\mu(x_i) - \mu(x_0) = 0$$

The algorithm considers training patterns as vectors in a multidimensional space, which it partitions in regions by locations and values of the samples. A class c is assigned to a random point of the feature space if it is the most frequent among the k nearest neighbors/training patterns of the location. Euclidean distance is often employed to decide the distance between locations in the feature space.

IV. THE EVOLUTIONARY PROCESS

The first goal that this research achieves is the prototyping of meta – data production from time series data, through the implementation of an innovative genetic algorithm. Roughly, the algorithm initiates by producing a random population of ‘trainers’ in its initial generation. These are nothing else than binary vectors, which are considered as chromosomes behaving in certain predefined ways and able to appropriately manipulate the initial raw time series. After the testing phase, each trainer is assigned a ‘fitness’ value, i.e. the root mean square error of the neural network trained and tested with meta-data produced by the said trainer/chromosome. Thereinafter, the next and subsequent generations of the algorithm are formulated according to a selection policy which elects the fittest members of the previous generation of trainers. It is obvious that the most potent and powerful trainers are the ones with the lowest RMS assigned to them, thus, the algorithm after a number of generations is expected to create a powerful neural network of an almost ideal RMS error.

With the following pseudocode the route of the evolutionary training of the network is shown:

```
Set the ideal RMS to reach
Set the trainer population
Set the recombination and mutation probabilities
Set the maximum training epochs

Change to the proper working directory
Read the initial train file
Read the initial test file
Select the time series train and test data
Split the train and test files to time series and non-time series data respectively

WHILE neuralAccuracy < ideal Accuracy
        Assembly the generation’s trainer population
        For eachindividual trainer:
            Map trainer chromosome to time series data
            Rejoin the derived meta – data to the saved non time series data
            Train and test the neural network
            Assign the derived neural RMS to the trainer chromosome
        Select trainers according to their fitness value for the next generation
        Apply recombination and mutation procedures to the selected trainer chromosomes
```

A. The Structure of the Trainer Population

The first and subsequent generations of the genetic algorithm start out by assembling a population of a user defined number of chromosomes, the genome of which consists of randomly chosen binary genes (0 or 1). Each chromosome is used so as to manipulate the input raw time series data according to their structure, in order to produce fitter educational meta data sets used in the supervised learning of the neural network. The genome of the chromosome is divided in two parts: the first part consists of a number of randomly chosen binary bits (0 or 1), equal to the time fragments of the initial raw time series data and is called the ‘activation part’ for it bears as its duty to carry out the instructions given by the behavioral mechanism. The second part of the chromosome, located at the beginning of the genome, is the ‘behavioral core mechanism’ of the chromosome. It consists of two supplementary random binary bits (0 or 1), the structure of which define a set of rules deciding the actions taken by the chromosome towards the raw data that it manipulates.

B. Data manipulation and chromosome mapping

Each training chromosome exists in order to ‘map’ its genes to the raw data set, according to the structure of its core mechanism. Thus, a number of meta data sets equal to the number of the user defined chromosomes for each generation of the algorithm is produced and is used for the training of the neural network. The ‘mapping’ of the chromosome onto the raw data set is essentially one of a number of descriptive statistics functions, chosen according to the following table 2:

Table 2. Representation of the mapping of chromosomes onto raw initial time series data, according to their core behavioral genes

Core Genes	**Description** **and meta-data example for time series** **44 32 17 8 8 12 1 18 30 48** **and chromosome genome** **1 0 0 1 1 0 0 1 0 1**
00	Discard All Zeros Resampling function, with which the raw time series will be stripped off of its values for which the corresponding genes of the trainer is 0 44 8 8 18 48
11	First One Last Zero Average Clustering function, which extracts the everage of the initial time series elements for every group of its own genes which start with the first 1 and end with the last zero 31 8 7 24 48
01	First-One-Last-Zero Median Clustering function, which returns the median of the initial data time series elements for each cluster which corresponds to the first 1 and the last zero of its own genes 32 8 8 24 48
10	First One Last Zero MinMax Clustering function, which returns the distance of the maximum to the minimum value of every cluster which corresponds to the first 1 and the last zero of its own genes 27 8 11 12 48

C. The survival of the 'fittest'

Each meta data set produced is used in supervised learning, passing through the neural network for which the RMS error is calculated and later assigned to each corresponding chromosome. Thus, the rating of the chromosome by its performance becomes a crucial factor for its survival. The policy of selecting the best offspring incorporates the stochastic procedure known as roulette wheel selection.

Let $f(\omega_i)$ denote the fitness function of each of $\omega = N$ individual chromosomes. Then, the chromosome ω_i, i = 1, 2, …, N has probability

$$\frac{f(\omega_i)}{\sum_{i=1}^{N} f(\omega_i)}$$

of being selected.

The roulette wheel incorporates a fitness proportionate selection operator, which elects to perpetuate the fittest chromosomes, i.e. the ones with the higher fitness score, in our case the ones with the lowest RMS errors. In this context, chromosomes with relatively high fitness scores are less likely to be eliminated. On the other hand, less fit chromosomes may not be extinguished from the genetic pool of the next generations. This results in the fact that some weaker solutions to the problem at hand may survive the algorithm sweep for the forming of the next generations, conveying their potentially useful genes to their offspring.

The next and subsequent generations of the algorithm are assembled after the fittest offspring have passed through a mutation and cross-over procedure according to a predefined probability respectively.

V. Results

In order to assess the robustness of the proposed evolutionary solution, it had to be compared towards a range of widely used classifiers, which were chosen among the most common in machine learning. For each classification method, a single metric was estimated, namely the accuracy of the classifier, which was calculated as the effective generalization of each one, expressed as the ratio of correctly classified input patterns to the total number of presented patterns during the testing phase. The accuracy metric, as defined above, was calculated using two testing techniques, namely the 70/30 data set random re-sampling and the area under Receiver Operating Characteristic(ROC) curve, using for both the 10 fold cross validation scheme. The following Table 3 depicts the results.

Table 3. Average classification accuracies among three classifiers and the evolutionary neural network for virus identification time series data

	Evolution Neural	**Naive Bayes**	**Class Tree**	**k -NN**
Accuracy	0.93	0.70	0.88	0.88
AROC	0.96	0.66	0.88	0.96

The outcome of the comparison yields that by applying an evolutionary resampling method on the time series raw data, in order to produce genetically enhanced meta data, we are able to control their dimensionality, achieving to produce a more robust classifier with better generalization potential. Also, although the time of training is much longer, the proposed classification system exhibits the same classification time demands as the rest of the classifiers, once trained and, of course is very competitive to the time an experts needs to make a decision by evaluating a signature curve.

VI. Conclusions

Genetic algorithms and neural networks are exciting technologies in the fields of artificial intelligence, machine learning and data mining. Numerous are the research works binding their powers, thus providing flexible, adaptable, swift and potential systems successfully applied on classification or prediction problems. Genetic algorithms are considered as modules to ‘breed’ solutions for optimization or prediction problems by means of simulated evolution, while neural networks are data modelling systems comprised of interconnected layered nodes which learn by example and are able to represent complex input to output relationships. In many cases the evolutionary process is entangled in the design of the structure of the network optimizing its architecture, in other applications it optimizes the training weight vector of a fixed architecture and in yet other references it manipulates both the structure and the training simultaneously. In this work we present and test an evolutionary neural network for the detection of plant viruses. The method producing the data for the system is the Bioelectric Recognition Assay (BERA) method for the detection of viruses. BERA uses special sensor electrodes containing certain reagents suspended in a gel matrix which, while interacting with the virus particles, produce electrical signals measured as a voltage. This interaction between the sensor antibodies and the virus particles lasts for a certain period of time, thus the voltage series produced are essentially time series data which are considered as signatures, each of which is a characteristic of the virus. The method proposed in this paper incorporates an evolutionary technique to manipulate these time series in the training phase of the neural object so as to produce genetically enhanced meta - data, in perspective of controlling the dimensionality of the input space. After the evolutionary neural object has been designed, trained and tested, it is put to contest against some of the most common classifiers in machine learning, namely the naive Bayesian classifier, the decision tree classifier and the k-nearest neighbors one. The whole method is essentially an attempt towards the development of a more credible classification system, exploiting the fascinating features of neural networks and genetic algorithms in the line of identifying viruses with the use of a technique, the products of which could up to now be assessed only empirically.

In our work the classification demand was supported by the biosensor method BERA and was applied to the identification of serious plant viruses, namely the tobacco rattle virus (TRV) and the cucumber green mottle mosaic virus (CGMMV). The main drawback of the traditional identification process was the empirical way that was employed in order to assess the results. The produced meta data set was used to train an evolutionary MLP, in the sense that it adapted its input layer to the provided meta data. This was further compared to other classifiers trained with the same initial raw time series data. The results show that the proposed technique yields better results in the testing phase of our classifier, demanding the same classification times, once trained. By applying an evolving genetic data clustering procedure on the initial data set, we were able to better control the dimensionality of the input data space on one hand and also to overcome the drawbacks of the empirical assessment on the other.

REFERENCES

[1]. A. Abraham, “Meta learning evolutionary artificial neural networks”, *Neurocomputing* 56 (2004) 1–38.

[2]. E. Alpaydin, “Techniques for combining multiple learners” in *Proceedings of Engineering of Intelligent Systems*, volume 2, pages 6-12, ICSC Press, 1998.

[3]. R. K. Agrawal and J. K. Singh, “Application of a Genetic Algorithm in the Development and Optimisation of a Non-linear Dynamic Runoff Model”, *Biosystems Engineering* (2003) 86 (1), 87–95.

[4]. F. Anctil, N. Lauzon, V. Andreassian, L. Oudin and C. Perrin, “Improvement of rainfall-runoff forecasts through mean areal rainfall optimization”, *Journal of Hydrology* (2006) 328, 717–725.

[5]. S. Avramidis and L. Iliadis, “Wood-Water Isotherm Prediction with Artificial Neural Networks: a Preliminary Study” *Holzforschung*, ISSN: 0018-3830 59 (3), 336-341. Berlin, New York, Walter De Gruyter & Co.

[6]. L. Bodri and V. Cermak, “Prediction of extreme precipitation using a neural network: application to summer flood occurrence in Moravia”, *Advances in Engineering Software* 31 (2000) 311–321.

[7]. L. Boillereaux, C. Cadet and A. Le Bail, “Thermal properties estimation during thawing via real time neural network learning”, *J. Food Eng*. 57(1) (2003) 17-23.

[8]. X. Cai, D. C. McKinney and L. S. Lasdon, “Solving nonlinear water management models using a combined genetic algorithm and linear programming approach”, *Advances in Water Resources* 24 (2001) 667–676.

[9]. C. L. Chang, S. L. Lo and S. L. Yu, “Applying fuzzy theory and genetic algorithm to interpolate precipitation”, *Journal of Hydrology* 314 (2005) 92–104.

[10]. K. W. Chau, “A split-step particle swarm optimization algorithm in river stage forecasting”, *Journal of Hydrology* 346 (2007), 131–135.

[11]. C. T. Cheng, C. P. Ou and K. W. Chau, “Combining a fuzzy optimal model with a genetic algorithm to solve multi-objective rainfall – runoff model calibration”, *Journal of Hydrology* 268 (2002) 72–86.

[12]. D. A. Elizondo, R. Birkenhead, M. Gongora, E. Taillard and P. Luyima, “Analysis and test of efficient methods for building recursive deterministic perceptron neural networks”, *Neural Networks* 20 (2007) 1095–1108.

[13]. A. J. P. Filho and C. C. dos Santos, “Modeling a densely urbanized watershed with an artificial neural network, weather radar and telemetric data”, *Journal of Hydrology* 317 (2006) 31–48.

[14]. P. J. Gallant and G. J. M. Aitken, “Genetic algorithm design of complexity-controlled time-series predictors”, 0-7803-8178-5/03, *IEEE XIII Workshop on Neural Networks for Signal Processing*.

[15]. T. Glezakos, T. Tsiligiridis, C. Yialouris, F. Maris, K. Ferentinos, “Feature Extraction for Time Series Data: an Artificial Neural Network Evolutionary Training Model for the Management of Mountainous Watersheds”. *EANN 2007, 10th International Conference on Engineering Applications of Neural Networks*, 29-31 August 2007, Thessaloniki, Hellas

[16]. J. V. Hansen, J. B. McDonald and R. D. Nelson, “Time series prediction with genetic algorithm designed neural networks: an Empirical comparison with modern statistical models”, *Computational Intelligence*, 15(3) (1999).

[17]. C. Harpham and C. W. Dawson, “The effect of different basis functions on a radial basis function network for time series prediction: A comparative study”, *Neurocomputing* 69 (2006) 2161–2170.

[18]. S. Haykin, *Neural Networks, A Comprehensive Foundation*, Prentice Hall International, New Jersey, 1989.

[19]. J. Demsar, B. Zupan and G. Leban (2004) “Orange: From Experimental Machine Learning to Interactive Data Mining”, White Paper (www.ailab.si/orange), Faculty of Computer and Information Science, University of Ljubljana.

[20]. A. Jain and A. M. Kumar, “Hybrid neural network models for hydrologic time series forecasting”, *Applied Soft Computing* 7 (2007) 585–592.

[21]. T. Kerh, and C. S. Lee, “Neural networks forecasting of flood discharge at an unmeasured station using river upstream information”, *Advances in Engineering Software* 37 (2006) 533–543.

[22]. S. Kintzios, E. Pistola, P. Panagiotopoulos, M. Bomsel, N. Alexandropoulos, F. Bem, I. Biselis, R. Levin, “Bioelectric recognition assay (BERA)”, *Biosensors and Bioelectronics* (16) 325 - 336, 2001.

[23]. S. Kintzios, E. Pistola, J. Konstas, F. Bem, T. Matakiadis, N. Alexandropoulos, I. Biselis, R. Levin, “Application of the Bioelectric recognition assay (BERA) for the detection of human and plant viruses: definition of operational parameters”, *Biosensors and Bioelectronics* 16: 467-480, 2001.

[24]. S. Kintzios, F. Bem, O. Mangana, K. Nomikou, P. Markoulatos, N. Alexandropoulos, C. Fasseas, V. Arakelyan, A-L. Petrou, K. Soukouli, G. Moschopoulou, C. Yialouris, A. Simonian, “Study on the mechanism of Bioelectric Recognition Assay: evidence for immobilized cell membrane interactions with viral fragments”, *Biosensors & Bioelectronics* 20: 907-916, 2004.

[25]. S. Kintzios, Ef. Makrygianni, E. Pistola, P. Panagiotopoulos, G. Economou, “Effect of amino acids and amino acid analogues on the in vitro expression of glyphosate tolerance in johnsongrass (Sorghum halepense L. pers.)”, *J. Food, Agriculture and Environment* 3: 180-184, 2003.

[26]. S. Kintzios, J. Goldstein, A. Perdikaris, G. Moschopoulou, I. Marinopoulou, , O. Mangana, K. Nomikou, I. Papanastasiou, A-L. Petrou, V. Arakelyan, A. Economou, A. Simonian, “The BERA Diagnostic System: An all-purpose cell biosensor for the 21th Century”, *5th Biodetection Conference*, Baltimore, MD, USA, 9-10/06/05, 2005.

[27]. L. Kuncheva, “Combining Classifiers by Clustering, Selection and Decision Templates. Technical report”, University of Wales, UK, 2000.

[28]. F. Liu, G. S. Ng and C. Quek, “RLDDE: A novel reinforcement learning-based dimension and delay estimator for neural networks in time series prediction”, *Neurocomputing* 70 (2007) 1331-1341

[29]. J. R. Ni and A. Xue, “Application of artificial neural network to the rapid feed back of potential ecological risk in flood diversion zone”, *Engineering Applications of Artificial Intelligence* 16 (2003) 105–119.

[30]. H. Niska, T. Hiltunen, A. Karppinen, J. Ruuskanen and M. Kolehmainen, “Evolving the neural network model for forecasting air pollution time series”, *Engineering Applications of Artificial Intelligence* 17 (2004) 159–167.

[31]. R. B. C. Prudencio, and T. B. Ludermir, “Meta-learning approaches to selecting time series models”, *Neurocomputing* 61 (2004) 121 – 137.

[32]. L. Pulido – Calvo, and M. M. Portela, “Application of neural approaches to one-step daily flow forecasting in Portuguese watersheds”, *Journal of Hydrology* (2007) 332, 1– 15.

[33]. F. Rossi, N. Delannay, B. Conan-Guez and M. Verleysen, “Representation of functional data in neural networks”, *Neurocomputing* 64 (2005) 183–210,

[34]. G. B. Sahoo, C. Ray and E. H. De Carlo, “Use of neural network to predict flash flood and attendant water qualities of a mountainous stream on Oahu, Hawaii”, *Journal of Hydrology* (2006) 327, 525–538.

[35]. R. K. Sivagaminathan and S. Ramakrishnan, “A hybrid approach for feature subset selection using neural networks and ant colony optimization”, *Expert Systems with Applications* 33 (2007) 49–60

[36]. E. Toth, A. Brath and A. Montanari, “Comparison of short-term rainfall prediction models for real-time flood forecasting”, *Journal of Hydrology* 239 (2000) 132–147.

[37]. Y. Wei, W. Xu, Y. Fan and H. T. Tasi, “Artificial neural network based predictive method for flood disaster”, *Computers and Industrial Engineering* 42 (2002) 383–390.

[38]. R. N. Yadav, P. K. Kalra and J. John, “Time series prediction with single multiplicative neuron model”, *Applied Soft Computing* 7 (2007) 1157–1163.

[39]. P. C. Young, “Advances in real-time flood forecasting”, *Philosophical Transactions of the Royal Society London. A* 360 (2002) 1433-1450