Genetic Programming and Serial Processing for Time Series Classification

Eva Alfaro-Cid evalfaro@iti.upv.es
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022, Valencia, Spain

Ken Sharman ken@iti.upv.es
Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022, Valencia, Spain

Anna I. Esparcia-Alcázar aesparcia@s2grupo.es
S2 Grupo, c/Ramiro de Maeztu 7, 46022, Valencia, Spain

doi:10.1162/EVCO_a_00110

Abstract

This work describes an approach devised by the authors for time series classification. In our approach genetic programming is used in combination with a serial processing of data, where the last output is the result of the classification. The use of genetic programming for classification, although still a field where more research in needed, is not new. However, the application of genetic programming to classification tasks is normally done by considering the input data as a feature vector. That is, to the best of our knowledge, there are not examples in the genetic programming literature of approaches where the time series data are processed serially and the last output is considered as the classification result. The serial processing approach presented here fills a gap in the existing literature. This approach was tested in three different problems. Two of them are real world problems whose data were gathered for online or conference competitions. As there are published results of these two problems this gives us the chance to compare the performance of our approach against top performing methods. The serial processing of data in combination with genetic programming obtained competitive results in both competitions, showing its potential for solving time series classification problems. The main advantage of our serial processing approach is that it can easily handle very large datasets.

Keywords

Classification, time series, genetic programming, serial data processing, real world applications.

1 Introduction

Abstractly, the problem tackled in this paper amounts to classification of finite length data sequences, as opposed to the more commonly encountered problem of classification based on feature vectors. That is, our data will consist of a sequence of observations over time (we call such a sequence a time series) as opposed to cross sectional data (feature vectors) where all observations are collected together and presented simultaneously (in parallel) to the classifier. There are many approaches for solving classification problems: statistical methods, neural networks, fuzzy logic, and so on. Recently, some researchers have reported success using genetic programming (GP) techniques for classification purposes. One of the advantages of this approach is that it is not constrained by

Manuscript received: February 14, 2012; revised: June 21, 2013; accepted: September 9, 2013.

a priori choice of a classification method. Rather, the classification algorithm is allowed to evolve to best solve the underlying problem at hand.

GP is an optimization method inspired by biological evolution that aims to find solutions that perform a user-defined task (Koza, 1992; Poli et al., 2008; Poli, 2010; Langdon et al., 2010). Unlike the classical optimization methods, which are based on tracking a certain trajectory, GP works with a population of candidate solutions (called individuals).

As for any evolutionary optimization technique, in GP an initial population of individuals is created at random. Each individual in the population is evaluated using a so-called fitness function. The fitness of an individual measures how well it solves the problem at hand, in our case how well the function classifies the time series data. The individuals that perform better in the evaluation process have a higher probability of being selected as parents for the new population. A new population is created using selection, crossover, and mutation operators. The individuals of this new population typically show better performance than those of the previous one, since the best individuals have a better chance of being selected for reproduction. The loop is run until a certain termination criterion is met, for example, obtaining near optimum solutions or when a predetermined finite number of generations is reached.

As opposed to other evolutionary techniques, the individuals being evolved in GP are functions and can be represented by a tree of no predefined size. This makes it very appropriate for classification and regression problems. In addition, it offers an escape from the black box techniques, providing an explicit function as a result.

In the literature, examples can be found of applications of GP to classification problems of feature vectors (for a review, see Jabeen and Baig, 2010). In some contributions GP evolves the classifier structure itself. For example, in Eggermont et al. (1999) different variations of GP applied on two-class classification problems are reported. In Liu and Khoshgoftaar (2004) a random sampling technique is introduced to reduce overfitting. In Kishore et al. (2000) the GP paradigm is extended for multi-class problems and the same authors go one step further in Kishore et al. (2001) integrating the GP classifier with feature space partitioning. Following the same trend, Muni et al. (2006) present an approach for multi-class classification problems that simultaneously optimizes the subset of features used and the classifier. Finally, in Oltean and Diosan (2009) a system called genetic programming–autonomous solver is presented and successfully tested on classification problems. That is, the role that GP plays in the classification process is that of evolving/generating a classification algorithm.

However, GP is usually used for optimizing a classifier structure generated by means of another method. For instance, in Dasgupta and Gonzalez (2001) a linear genetic tree is proposed in order to evolve complex fuzzy rule sets for solving classification problems. In Tsakonas (2006) four structures, namely, decision trees, fuzzy rule-based systems, feedforward neural networks, and fuzzy Petri-nets, are evolved using GP.

In other works, GP has been used as a preprocessing tool to evolve projections that translate the data into a new vector space where projected data can be more easily classified. For instance, in Estébanez et al. (2007) the data are projected into a 3D space and then they are classified using a simple linear perceptron. Other approaches use GP for a feature extraction stage prior to classification (Eads et al., 2002; Otero et al., 2003).

As mentioned previously, all these approaches classify feature vectors. There are some works that deal with the problem of time series classification (Estébanez et al., 2007; Eads et al., 2002) but again they treat the time series as a feature vector.

In the field of digital signal processing there are a few references where the authors try to circumvent the inherent difficulty (due to the data handling limitations) of working with time series in GP. In Holladay and Robbins (2007) a new vector-based GP language is developed, where vectors can be treated as single input data, instead of being handled element by element. The main disadvantage of this approach is the added complexity in the data handling, given that all data, including simple types, need to be treated as a structure. Also, in Esparcia-Alcázar (1998) delay nodes are introduced as a mechanism for handling data series in a channel equalization problem.

In the field of GP forecasting of time series, on the other hand, there is a great deal of literature available, mainly on financial series (Kaboudan, 2000; Hui, 2003; Ahalpara and Parikh, 2006; Borrelli et al., 2006).

The aim of this paper is to present a new approach that considers a sequential treatment of the time series to classify, instead of considering a parallel approach where all the data are treated simultaneously. The serial processing of data presented here preserves the essence of the time series by processing the elements in the same sequential order that they are measured. Also, every time one element of the sequence is fed into the classifier, an output is generated. This output is fed back as an extra input into the classifier. The idea is similar to that of a window sliding along the time series. There will be as many classification steps as elements in the time series to classify and the output of the last step will be used as a classification result. Some authors have used a sliding window for time series prediction (Hui, 2003; Wagner and Michalewicz, 2008), although their approaches did not include any output feedback. The following equations show the sliding window effect considering a window size 8:

$$\begin{aligned} \text{Step } n \quad & \ldots \underbrace{x_{n\text{-}7}\ x_{n\text{-}6}\ x_{n\text{-}5}\ x_{n\text{-}4}\ x_{n\text{-}3}\ x_{n\text{-}2}\ x_{n\text{-}1}\ x_n}\ x_{n+1} \ldots \\ \text{Step } n+1 \quad & \ldots\ x_{n\text{-}7}\ \underbrace{x_{n\text{-}6}\ x_{n\text{-}5}\ x_{n\text{-}4}\ x_{n\text{-}3}\ x_{n\text{-}2}\ x_{n\text{-}1}\ x_n\ x_{n+1}} \ldots \end{aligned} \tag{1}$$

A preliminary approach to this idea was already presented in Alfaro-Cid, Sharman et al. (2006). The objective of this work was to evolve a learning machine using a GP that incorporates in its function set what we called a learning node. Such a node was tuned by a second optimization algorithm, mimicking a natural learning process and providing the GP tree with added flexibility and adaptability. These nodes consist of a value α, which is variable, and one input. Given an input x, the output, y, of the node is equal to $y = \alpha \cdot x$. If learning nodes are included in the function set, we are effectively adding variable gains at certain points of the system. Varying the value of these gains implies changing the way the system responds.The result of the evolution is a system with a fixed structure but with some variable parameters. The system could then learn new tasks in new environments without undergoing further evolution, just by tuning the α parameters. In this context the learning machine was tested using serial processing of data for a toy problem: the two-way classification of stochastic time series data values.

In this paper we aim to more exhaustively test the serial processing approach in three problems: the classification of white noise signals that have been filtered through band pass filters with two different passband frequencies, the classification of electroencephalogram (EEG) signals from a brain-computer interface (BCI) and the classification of data samples from an automotive subsystem. The latter two problems are real-world applications and the datasets used were gathered for classification competitions. This gives us the chance to compare the results provided using GP with serial processing against the results obtained by other researchers.

Thus, summarizing, the original contribution of this paper lies in the combination of GP and a serial processing of data for time series classification. Instead of considering the time series as a feature vector and treating all the elements in the series in parallel, the elements are processed serially and the last output of the process is considered as the classification result. This allows for the handling of very large datasets. On the other hand, since the classifier is evaluated as many times as elements there are in the time series, the evolution of the classifier is slower.

The layout of the paper is as follows: Section 2 describes the GP algorithm and how it was applied to the classification problem. The emphasis is placed on the serial processing of the data. Section 3 explains the experimental procedure followed in the work. The following three sections, Section 4, Section 5 and Section 6, describe the problems solved with our approach, the data used and the particular GP implementation applied, as well as the results obtained. Finally, in Section 7 some conclusions are drawn.

2 Genetic Programming for Classification

Genetic programming (Koza, 1992; Poli et al., 2008) is an evolutionary computation technique based on the idea that in nature structure undergoes adaptation. The structure created over a period of time is the outcome of natural selection and sexual reproduction. Thus, GP is a structural optimization technique (as opposed to a parametric optimization technique).

In the GP algorithm, traditionally, the structure being evolved has a tree shape. That means that the size and shape of the solutions are not defined a priori as in other methods from the field of evolutionary computation, but they evolve along the generations.

Prior to creating a GP environment, the designer must define which functions (internal tree nodes) and terminals (leaf branches) are relevant for the problem to solve. This choice defines the search space for the problem in question. Given a fairly small set of functions, different combinations of them allow the GP to specify a huge range of multivariable functions. On the other hand, increasing the number of functions increases the size of the search space. Therefore, the inclusion of too many functions may hamper the efficiency of the search algorithm.

Once the function and terminal sets are chosen, the first population needs to be initialized. The initialization process is more complex for GP than for other evolutionary algorithms due to the increased complexity of the individuals' representation. In his pioneering work, Koza (1992) presented three methods of initializing the population: *full method*, *grow method*, and *ramped half-half*. The full method involves creating trees with a fixed length between the root and every terminal node. The grow method involves creating trees with a specified maximum length between the root and every terminal node. Thus, the shapes of the trees in the population show a bigger diversity. Finally, the ramped half-half method creates an equal number of trees for every length value from two up to the maximum. For each length value, half of them are created using the full method and the other half using the grow method. Koza (1992) recommends the use of ramped half-half because of the wide variety of shapes it creates. According to Koza's suggestions, in this work, ramped half-half was used as the initialization method.

Once an initial population of trees is created at random, each individual in the population is evaluated using a fitness function. This fitness function assigns a value to the individual according to how well it solves the problem at hand (i.e., the classification of the training set of data). Then, a new population is created through reproduction, crossover, and mutation. Generally speaking, individuals with a higher level of fitness

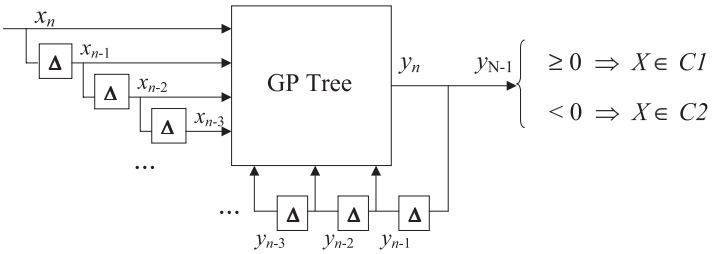

Figure 1: Serial processing of data using a single input which is applied N times. Note that $X = \{x_n, n = 0, \cdots N-1\}$.

are selected for reproduction mimicking a "survival of the fittest" strategy. However, occasionally lower fitness individuals are selected to maintain population diversity. Some of these individuals are chosen for crossover and mutation with a certain probability. The loop is run until a certain termination criterion is met.

One of the problems encountered by GP researchers is that of code bloat (Luke, 2000; Luke and Panait, 2006). Bloat occurs when the trees grow in size without an improvement in fitness. This flaw causes a waste of computer resources evaluating huge trees and difficulties in the understanding of the final solutions. It spoils the convergence by hampering the modification of trees in a meaningful way.

2.1 Serial Processing of Data

In a conventional classifier, the data to be classified are normally presented to the system as a vector, that is, all the data are presented simultaneously to the classifier. In our case, we adopted an alternative approach in which the data are presented to the classifier in series. The aim of this is to reduce the complexity of the classifier. A conventional classifier for data with N features requires a system with N inputs. In our approach we have a single input which is applied N times. In effect we are trading off space for time.

To do this, our GP has a set of terminals $\{x'_0, x'_1, \ldots, x'_M\}$ which represent the current input and the previous M inputs to the classifier. That is, at time step n, the $m+1$ terminals are, $x'_0 = x_n, x'_1 = x_{n\text{-}1}, \ldots, x'_M = x_{n\text{-}M}$. We also have another set of terminals labeled $\{y'_1, y'_2, \ldots, y'_K\}$ representing K previous outputs from the GP tree at time step n. That is, $y'_1 = y_{n\text{-}1}, y'_2 = y_{n\text{-}2}, \ldots, y'_K = y_{n\text{-}K}$ (see Figure 1). The parameters M and K are user defined and will depend on the application at hand.

The upshot of all this is that our GP trees implement functions of the form:

$$y_n = f(x_n, x_{n\text{-}1}, \ldots, x_{n\text{-}M}, y_{n\text{-}1}, \ldots, y_{n\text{-}K}) \quad (2)$$
$$n = 0, 1, 2, \ldots, N.$$

In order to deal with the first part of the time series, the serial processing of the series starts on sample x_M so that the required previous inputs are available. The previous outputs are set to zero.

Note that the output of $f()$ is also a time series. The actual function $f()$ is, of course, defined by the structure of the GP tree.

The resulting functions are not unique since GP is a stochastic algorithm that may produce different results each time it is executed.

2.2 Signal Classification

In this work we are interested in a two-category classification of time series.

For the implementation, let the data to be classified be $X = \{x_0, \ldots, x_{N\text{-}1}\}$. Then we apply this dataset as input to a GP tree and calculate the outputs $y_i, i = 0, \ldots, N-1$ (i.e., the tree is executed N times in total). The final output, $y_{N\text{-}1}$, indicates the classification results according to:

$$y_{N\text{-}1} \geq 0, \; X \in C1 \tag{3}$$

$$y_{N\text{-}1} < 0, \; X \in C2 \tag{4}$$

where $C1$ and $C2$ are the two class labels.

During evolution, we use a training set of T independent time series and known labels $\{(X_i, \; c_i), \; i = 1, \ldots, \; T\}$ where $X_i \in \Re^N$ is the N-dimensional feature vector, and $c_i \in \{C1, C2\}$.

To evaluate the fitness of each individual, we apply each X_i in turn and calculate its chosen class, c_i', according to Equations (3) and (4). We then count the number of hits as the number of times $c_i = c_i'$. Note that to calculate the fitness of an individual, the individual's tree must be evaluated TN times.

3 Experimental Procedure

In order to test our serial processing approach, we chose three different problems that deal with two-category classification of time series. These problems were (I) classification of time series with different power spectral densities, (II) classification of states in a brain-computer interface (BCI) using electroencephalogram—EEG—signals, and (III) classification of data samples from an automotive fault finding application.

The time series used in the first classification problem were generated for the purpose of this work by filtering white noise signals with filters with different passbands. The EEG signals used in the classification problem presented here were gathered for the BCI Competition II (http://www.bbci.de/competition/ii/). The objective of the competition was to evaluate the current state of the art of the BCI field. The automotive signals were used in the Ford Classification Challenge (http://home.comcast.net/~nn_classification/), one of the competitions hosted at the IEEE World Congress on Computational Intelligence (WCCI) that took place in Hong Kong in June, 2008. The aim of the competitions was to stimulate research in computational intelligence and promote fair evaluations.

The use of data from online competitions in the case of the EEG problem and the Ford problem provides us with a good benchmark of results to compare our results against. That is, in each of these competitions a fairly large number of researchers submitted results obtained with a broad set of different classification techniques. This will allow us to validate the quality of our results against a range of different classification methods.

The following sections describe in detail the three problems considered, the GP implementation used for each problem, and the results obtained.

4 Problem I: Power Spectral Density Problem

The problem to solve is the classification of white noise signals filtered with filters that have different passbands (see Figure 2). The problem was solved for three different datasets. In the first one, the filters have separated passbands, and in the second case the passbands are very close, thus increasing the difficulty of the problem. Finally, the third set considers filters with overlapping passbands.

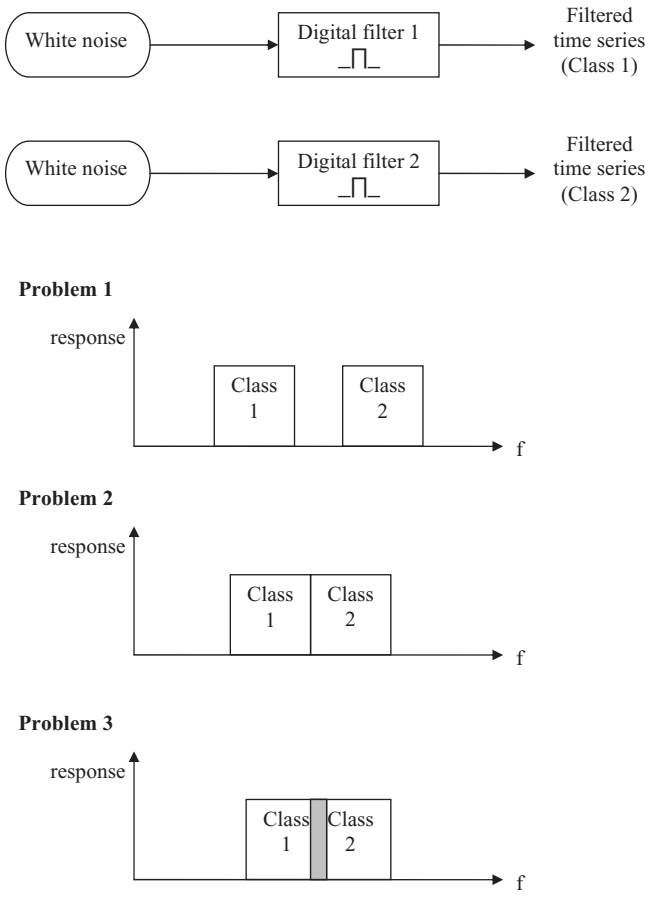

Figure 2: Generation of time series with different power spectral densities.

4.1 Description of the Data

In this problem we worked with three different sets of data. Each of them was generated from 200 white noise signals. Half of them were passed through a passband filter with a certain passband, while the other half were passed through a passband filter with a different passband. Each white noise signal is a sequence of 100 elements.

Therefore, the classification problems under consideration are:

- Problem 1: Classification of two time series with different power spectral densities.

  Series 1: White noise passed through a band pass digital filter, passband 0.1–0.2 (of the sampling frequency)

  Series 2: White noise passed through a band pass digital filter, passband 0.3–0.4 (of the sampling frequency)

- Problem 2: Classification of two time series with different (but closer) power spectral densities.

Series 1: White noise passed through a band pass digital filter, passband 0.18–0.2 (of the sampling frequency)

Series 2: White noise passed through a band pass digital filter, passband 0.21–0.3 (of the sampling frequency)

- Problem 3: Classification of two time series with overlapping power spectral densities.

Series 1: White noise passed through a band pass digital filter, passband 0.18–0.25 (of the sampling frequency)

Series 2: White noise passed through a band pass digital filter, passband 0.21–0.3 (of the sampling frequency)

The dataset consisted of 200 data records for each problem. Half of them were used for training the GP and the other half for testing the solutions.

4.2 GP Implementation

The GP implementation used is based on the JEO (Java evolving objects) library developed by Arenas, Dolin et al. (2002) within the European project DREAM (Distributed Resource Evolutionary Algorithm Machine; Arenas, Collet et al., 2002). The project's aim was to develop a complete distributed peer-to-peer environment for running evolutionary optimization applications.

JEO is a software package in Java, easily extendable, and integrated in DREAM. In the context of GP, the standard GP algorithm proposed by Koza (1992) is implemented in JEO as a default. JEO includes a genome structure with a tree shape and reproduction and crossover operators. Basically, the user only needs to implement the methods that are problem dependent, that is, fitness evaluation and construction of the function and terminal sets. In our case, the default algorithm was modified to use tournament selection and subtree mutation.

We used a population size of 500 individuals. The termination criterion was a 98% classification success. The crossover, mutation, and cloning rates were set at 0.8, 0.1, and 0.1, respectively.

The function set includes the four arithmetic operators as well as an *if* clause. The *if* clause takes four arguments: if $arg_1 \leq arg_2$ then return arg_3, else return arg_4. The division operator is protected so that if the denominator is zero, the division returns a zero value.

As explained in Section 2.1, the terminal set consists of $M + 1$ input terminals $\{x'_0, x'_1, \ldots, x'_M\}$ which represent the current input and the previous M inputs to the classifier and another set of K terminals labeled $\{y'_1, y'_2, \ldots, y'_K\}$ representing K previous outputs from the GP tree. For this particular problem $M = K = 9$. Thus, there are 19 terminals in the terminal set. This could resemble a lot but in a parallel approach we would need to use as many terminals as elements are in the data sequence, that is, 100 terminals for solving the same classification problem (for a time series of length 100 points).

4.3 Results

Table 1 shows the results obtained for the three problems of classification of two time series with different power spectral densities. For each problem, 50 runs of the GP algorithm were executed. The table shows the percentage of hits obtained during the

Table 1: Results obtained for the classification of two time series with different power spectral densities.

	Testing hits (%)
Problem 1	
Average	98.64
Standard deviation	2.59
Best	100
Problem 2	
Average	96.70
Standard deviation	3.97
Best	100
Problem 3	
Average	88.08
Standard deviation	8.63
Best	98

testing phase. It can be seen from the table that on increasing the difficulty of the problem, the average and best results worsen. In all runs, GP converged to a solution that obtains at least a 98% classification success, our termination criterion.

The best results from each of the 50 runs were compared as for the percentage of testing hits they produce. The following three equations represent the classifiers that obtained the highest percentage of testing hits for each problem under consideration:

$$\text{Problem 1: } y_n = y_{n\text{-}1} + x_{n\text{-}5}x_{n\text{-}8} \tag{5}$$

$$\text{Problem 2: } y_n = y_{n\text{-}6} + x_{n\text{-}1}x_{n\text{-}8} \tag{6}$$

$$\begin{aligned}\text{Problem 3: } y_n = {} & y_{n\text{-}2} + x_{n\text{-}3}\,(y_{n\text{-}8} + x_{n\text{-}3}\,(2x_{n\text{-}3}x_{n\text{-}9} + y_{n\text{-}7} - x_n x_{n\text{-}3}) \\ & + y_{n\text{-}2}\,(\text{if } x_{n\text{-}7} \leq x_n \text{ then } x_{n\text{-}6} \text{ else } x_{n\text{-}5}))\end{aligned} \tag{7}$$

As can be seen from Equations (5–7), the classifiers that scored the highest number of testing hits (100%) for Problems 1 and 2 present a very similar structure. However, the best classifier for Problem 3 is far more complicated. This could be expected given the increased problem complexity of Problem 3.

Let us analyze Equation (5). Substituting $y_{n\text{-}1}$ in Equation (5), an expression in terms of previous inputs and the output $y_{n\text{-}2}$ can be obtained: $y_n = y_{n\text{-}2} + x_{n\text{-}5}x_{n\text{-}8} + x_{n\text{-}6}x_{n\text{-}9}$. Repeating this process $n - 9$ times, we obtain the following expression for Equation (5):

$$y_n = y_8 + \sum_{k=0}^{n\text{-}9} x_{n\text{-}5\text{-}k} \cdot x_{n\text{-}8\text{-}k}. \tag{8}$$

This is a result of the fact that, in order to deal with the first part of the time series, the initial outputs (y_n where $n < M$) are set to zero. The expression could be simplified as:

$$y_n = \sum_{k=0}^{n\text{-}9} x_{n\text{-}5\text{-}k} \cdot x_{n\text{-}8\text{-}k}. \tag{9}$$

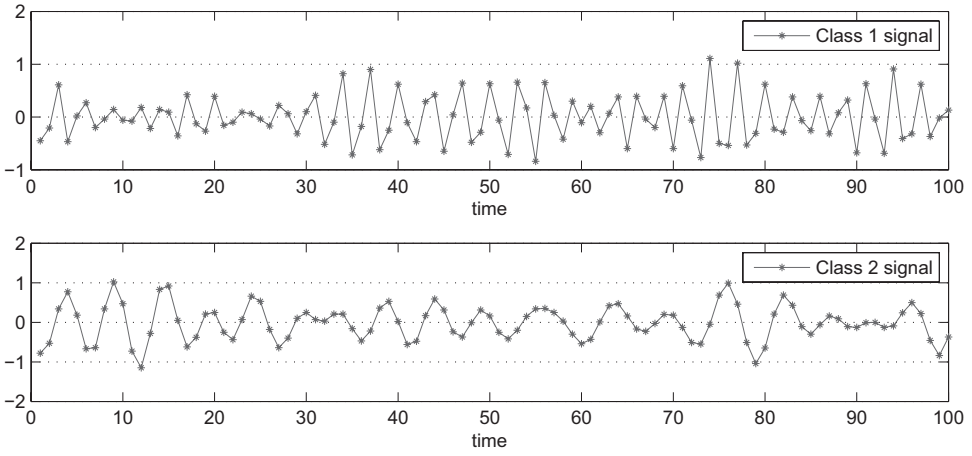

Figure 3: Examples of class 1 and 2 signals in the testing set of Problem 1.

The term $y_{n\text{-}1}$ in Equation (5) is used as a store that allows the building of the sum term in Equation (9). Therefore, $m = n - k$, and the resulting expression is:

$$y_n = \sum_{m=9}^{n} x_{m\text{-}5} \cdot x_{m\text{-}9} \tag{10}$$

Figure 3 shows two signals included in the testing set of Problem 1. One belongs to class 1 and the other to class 2. Due to the filtering through two band pass filters with different passband frequencies, the frequency of the cycles in both signals is different. That implies that the number of samples included in each cycle varies. The higher the frequency, the higher the number of cycles in the recorded period and the fewer the number of sample inputs in the cycle.

Figure 4 plots the time series of the product $(x_{m\text{-}5} \cdot x_{m\text{-}9})$ from $m = 9$ to $m = n$ for the signals of Figure 3. It can be observed that the time series is mostly positive for the class 1 signal and mostly negative for the class 2 signal. In class 1 signals, there are roughly three samples per cycle. That means that an element in the time series is usually multiplied by another element that takes a close value, since the cycles repeat themselves every three samples. The product of both samples is then positive. The addition of all these positive terms results in a positive value and, consequently, in a classification of the signal as class 1. On the other hand, in class 2 signals there are around five samples per cycle. Therefore, two elements within a distance of three are very likely to be of different sign, and the resulting product will have a negative value. The addition of all these negative terms results in a negative value and, consequently, in a classification of the signal as class 2.

A parallel analysis could be performed for Equation (6) in Problem 2. In this case the number of samples per cycle for class 1 signals is around seven, while in class 2 signals it is around 10. Therefore, the term $x_{m\text{-}1} \cdot x_{m\text{-}8}$, where the product of two samples with a delay of seven is calculated, will be mostly positive for class 1 signals, since in this case the number of samples per cycle is around seven, and mostly negative for class 2 signals.

It is interesting to note that the GP algorithm has identified the number of samples per cycle for one class series for each problem and has built a very simple, but

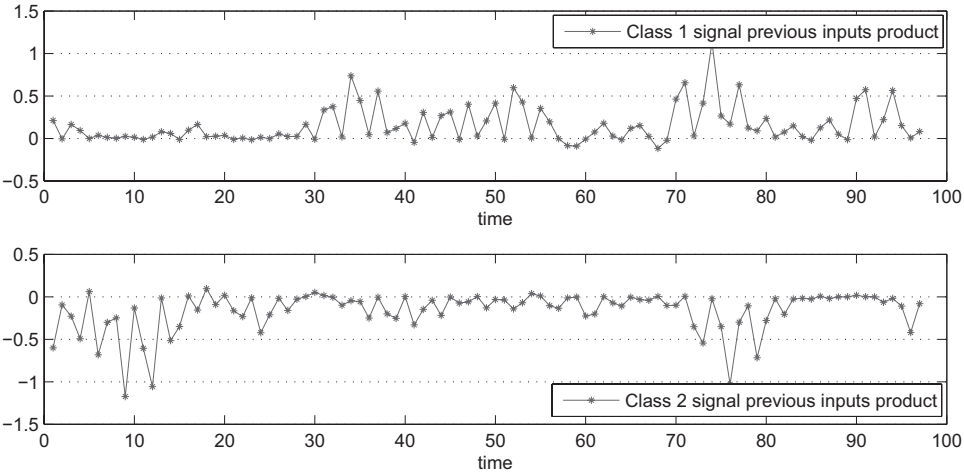

Figure 4: $(x_{m\text{-}5} \cdot x_{m\text{-}8})$ series for the signals in Figure 3.

effective, classification strategy. In Problem 3 this strategy does not work because of the overlapping passband frequencies, so GP has resorted to a more complex classifier.

5 Problem II: EEG Problem

BCI technology is based on the idea that different states of mind produce different EEG signals (Wolpaw et al., 2000). Therefore, it might be possible, by measuring and classifying the EEG signals, to translate certain states of mind into actions. The ultimate goal of the research in this field is to develop a system that will allow an individual to control an external device through his or her EEG signals.

In a BCI system, first, the EEG data are acquired. Then this data need to be preprocessed to eliminate noise and artifacts (e.g., eye blinking). Then, there is a stage of feature analysis where the EEG signals are compressed. Afterward, the data are divided into the training set and the test set. BCI systems need to be trained because every individual has different EEG patterns. In the training phase, the computer selects a mental state and the user focuses on whatever thought he or she associates with that state for a while. The EEG data are stored during that time, and they are also processed and classified. Finally, the user gets some feedback on the success or failure of the classification. Once the training is over, we hope that the BCI system will be able to classify the EEG readings with high accuracy.

5.1 Description of the Data

In this work we are interested in a two-category classification of the EEG data (Alfaro-Cid, Esparcia-Alcázar et al., 2006). In fact, we want to determine whether the user's state of mind is meant to be a command for moving a screen cursor up or down. The EEG signals used in the classification problem presented here were provided by the University of Tübingen (data sets Ia) for the BCI Competition II (Blankertz et al., 2004) (http://www.bbci.de/competition/ii/).

The datasets were taken from a healthy subject. The subject was asked to move a cursor up and down on a computer screen, while his cortical potentials were taken. Cortical positivity/negativity led to a downward/upward movement of the cursor. An

interval of 3.5 s of every trial is provided for training and testing. Using a sampling rate of 256 Hz results in 896 samples per channel for every trial. Samples of the six channels being recorded are stored concatenated. Therefore, the full raw data set is 5,376 samples for each trial.

The training set consisted of 268 data records ($T = 268$). After evolution, we tested the best evolved individuals on a set of 293 unseen data records.

5.2 GP Implementation

In this experiment we used ECJ (http://cs.gmu.edu/eclab/projects/ecj/), an evolutionary computation library in Java developed at George Mason University's ECLab (Evolutionary Computation Laboratory). The GP module in ECJ includes more features than JEO. Also, ECJ has a lively community of users contributing to the work and reporting bugs. As a consequence, the library is continuously improving.

We used a GP population of 1,000 individuals over 51 generations. The percentage of elitism was 1%. A new generation was created through subtree crossover (90% of the population) and cloning (10% of the population).

As a method of bloat control, we included a new crossover operator, bloat-control-crossover, that occurs with a probability of 0.4. This crossover operator implements a bloat control approach described in Alfaro-Cid et al. (2010) and inspired in the "prune and plant" strategy used in agriculture. It is used mainly for fruit trees and it consists of pruning some branches of trees and planting them in order to grow new trees. The idea is that the worst tree in a population will be substituted by branches pruned from one of the best trees and planted in its place. This way, the offspring trees will be of smaller size than the ancestors, effectively reducing bloat.

In this problem the terminal set consists of 41 terminals: $x_0^1, \cdots x_5^1, x_0^2, \cdots x_5^2, x_0^3, \cdots x_5^3, x_0^4, \cdots x_5^4, x_0^5, \cdots x_5^5, x_0^6, \cdots x_5^6, y_1, \cdots y_5$, where x_j^i is the value of the sequence j times prior to the current input for channel i. In the EEG case, we need six times the usual number of input terminals, since there are measures from six different channels. That is, the data sequence is the result of concatenating the measures of six channels. For classification purposes we are considering the six channels as six independent input data sequences, since they are independent time series measured simultaneously. Given that the number of terminals increases considerably because of this, for this problem we have set $M = K = 5$, which amounts to 41 terminals. For parallel processing of the data 5,376 terminals would be necessary.

As in Problem I, the function set includes the four arithmetic operators as well as an *if* clause. The *if* clause takes four arguments: if $arg_1 \leq arg_2$ then return arg_3, else return arg_4. The division operator is protected, so that if the denominator is zero, the division returns a zero value.

5.3 Results

Table 2 shows the results obtained for the EEG classification problem. Thirty runs of the GP algorithm were executed.

Figure 5 shows the percentile 10, 50, and 90 of the best solutions of each run along the generations. It can be seen how the improvement in the solutions slows down while the number of generations increases. To further test the convergence of the GP solution, a Wilcoxon signed ranked test was performed. The Wilcoxon signed ranked test is a nonparametric hypothesis test designed for use when the samples are measured on two occasions or under two different conditions. In this case an ANOVA test at different points of the evolution could not be performed due to the autocorrelation of the series.

Table 2: Results obtained for the classification of EEG signals.

	Testing hits (%)
Average	78.08
Standard dev.	8.20
Best	87.37

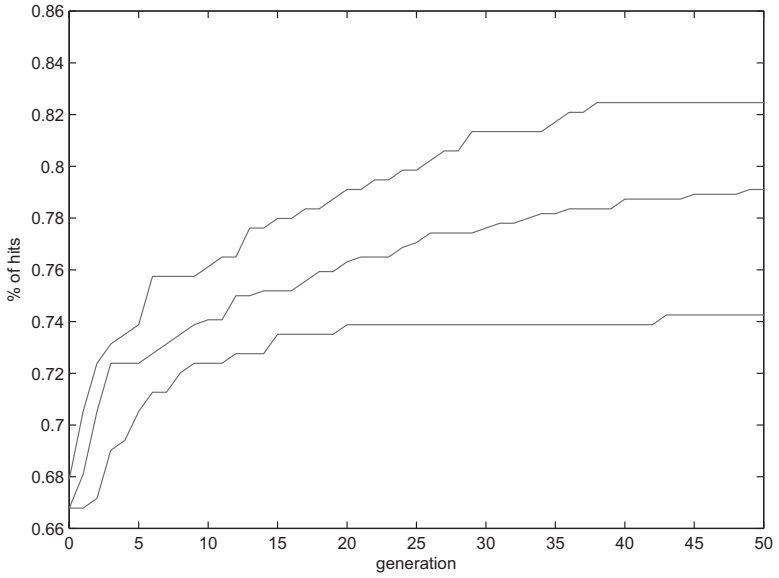

Figure 5: Evolution of the percentiles 10, 50, and 90 of the best solutions in the EEG runs.

Our aim is to ensure that by the time the evolution was stopped (that is, after 51 generations), the GP had converged. With this purpose, a Wilcoxon signed ranked test was performed with the mean values of each run in generation 51 and six generations before. The result of the test is that, with a significance level of 5% it can be concluded that the differences between the two scores are not statistically significant, that is, there was not significant improvement in the results in the last generations.

The best results from each of the 30 runs were compared as for the percentage of testing hits they produce. The following equation represents the classifier that obtained the highest percentage of testing hits for the problem at hand:

$$y_n = \begin{cases} y_{n\text{-}1} - x^1_{n\text{-}3}, & \text{if } y_{n\text{-}1} - x^2_{n\text{-}3} - 2x^1_{n\text{-}3} \leq y_{n\text{-}3} \\ y_{n\text{-}1} - x^1_{n\text{-}3} - x^2_{n\text{-}3}, & \text{if } (y_{n\text{-}1} - x^2_{n\text{-}3} - 2x^1_{n\text{-}3} > y_{n\text{-}3}) \text{ and } (x^2_{n\text{-}3} \leq y_{n\text{-}3}) \\ y_{n\text{-}1} - x^1_{n\text{-}3} - 2x^2_{n\text{-}3}, & \text{if } (y_{n\text{-}1} - x^2_{n\text{-}3} - 2x^1_{n\text{-}3} > y_{n\text{-}3}) \text{ and } (x^2_{n\text{-}3} > y_{n\text{-}3}) \end{cases} \quad (11)$$

As can be observed, Equation (11) only includes terminals with the supraindex 1 and 2. This indicates that the GP found that data from channels 1 and 2 are the most adequate for the EEG classification in this case. This is one of the advantages of obtaining explicit functions as results, in that they can be analyzed and valuable conclusions may be drawn.

Table 3: Results of "The BCI Competition II."

Rank	Error (%)	Research lab	Classification method
1	11.3	MIT	Discriminant analysis
2	11.6	Fraunhofer FIRST (IDA)	Regularized discriminant analysis
3	11.9	National Taiwan University	Support vector machine
4	15.0	National Taiwan University	Nonlinear support vector machine
5	15.7	University of Florida	Hidden Markov model with 10 states
6	17.1	Yale University	Support vector machine
7	17.4	University of Tübingen	Linear discriminant analysis
8	17.8		Regularized linear Fisher discriminant
9	19.1	University of Florida	Majority vote of different five methods
10	19.8	University of Florida	Recursive multi-layer perceptron
11	23.5	Sahand University of Technology	Neural network
12	24.6	Technical University of Graz	Multi-layer perceptron
13	34.5	University of Florida	Time-delay neural network predictor
14	46.8	University of Florida	Nonparametric Bayes classifier
15	49.1	University of Lille	Stochastic algorithm (GloBo)

This classification tree scored a 12.63% error in the testing phase. The results of the BCI Competition II can be found online (see http://www.bbci.de/competition/ii/results/index.html). Fifteen research teams from various nationalities participated in the competition. The percentages of error obtained ranked from 11.3% to 49.1%. There were a variety of classification methods proposed, among them, support vector machines, neural networks, and discriminant analysis. Table 3 shows the results obtained for each of the participants of the BCI competition.

As can be seen from Table 3, the result of the proposed GP algorithm would be classified between the 3rd and 4th position. The three candidates that qualified first obtained error percentages of 11.3%, 11.6%, and 11.9%, respectively. The two contributions that qualified first and second used discriminant analysis and the contribution that qualified third used support vector machines.

In the first qualified approach, each signal was represented by four feature values that were fed, subsequently, into the discriminant analysis classifier. The feature vectors found to be more discriminant in the analysis performed in Mensh et al. (2004) were time domain features of the slow cortical potentials for channels 1 and 2 and frequency-domain features for channels 4 and 6. It is interested to note that channels 1 and 2 were identified as the most discriminant in the time domain. This is consistent with the results obtained by the GP approach, that works in the time-domain (serial-feeding of data) and only considers channels 1 and 2 for classifying the signals.

In Mensh et al. (2004) the best results in classification are obtained using all four features (11.3% testing error). The classification performed using only time-domain features for channels 1 and 2 obtained a higher error (17.4% testing error). However, when only the frequency domain features were used, the classification error increased significantly. That means that, according to Mensh et al. (2004), although the time domain features are the most discriminant when classifying the EEG data, there is also a frequency component that has an effect. The GP, given the serial processing of data used, is missing that information, and that could justify the differences in performance between both methods.

The contributor that qualified fourth obtained an error percentage of 15.0% (worse than the error obtained with the proposed GP algorithm) using a nonlinear support vector machine. The remaining contributors used a variety of approaches, among them, neural networks.

All contributors have tackled the classification problem as a feature classification problem. In order to reduce the number of features (i.e., samples) in the signal to classify, groups have resorted to preanalysis of the data either to find some relevant features that describe the signal or to downsample it. In fact, some authors pointed out that feature selection had become the main challenge of the competition.

The proposed GP approach has obtained good results with no preprocessing of data and no need of expert knowledge on EEG waves. In fact, the GP algorithm converged to a solution that focused on the two channels identified by the first qualified contributor as more relevant in the time-domain without preanalyzing the data.

6 Problem III: Ford Problem

The Ford Classification Challenge competition was organized by Dr. Mahmoud Abou-Nasr and hosted at the IEEE World Congress on Computational Intelligence 2008 (http://home.comcast.net/~nn_classification/). The challenge problem was motivated by a potential automotive application. The results presented here relate to data set Ford_A.

6.1 Description of the Data

Data samples from a Ford automotive subsystem were collected in batches of 500 samples per diagnostic session. The objective is a classifier that will determine whether a certain symptom exists or does not exist. The 500 samples collected in each diagnostic session represent a set of sequential values of the measured variable, where sample $n + 1$ occurs after sample n, that is, they form a finite data sequence, not a feature vector. Also, the beginning of the sampling process is not aligned with any external circumstance or any aspect of the observed pattern and the data were collected in typical operating conditions, with minimal noise contamination.

The dataset was divided into training, validation, and testing data. The training data used for training the classifier consists of 3,271 training patterns. The validation data consist of 330 patterns, and the testing data consist of 1,320 patterns.

6.2 GP Implementation

The GP implementation for the Ford problem is very similar to the one used for the EEG classification. It was also implemented in ECJ.

A population of 1,000 individuals is evolved over 51 generations. The percentage of elitism is 1%. A new generation is created through subtree crossover (40% of the population), bloat-control-crossover (40% of the population), subtree mutation (10% of the population), and cloning (10% of the population).

For this particular problem, $M = K = 9$. Therefore, there are 19 terminals in the terminal set. In a parallel processing of the data the number of terminals would be 500.

As in problems I and II, the function set includes the four arithmetic operators as well as an *if* clause. The *if* clause takes four arguments: if $arg_1 \leq arg_2$ then return arg_3, else return arg_4. The division operator is protected so that if the denominator is zero, the division returns a zero value.

Table 4: Results obtained for the classification of Ford signals.

Validation hits (%)	Testing hits (%)
99.40	99.60

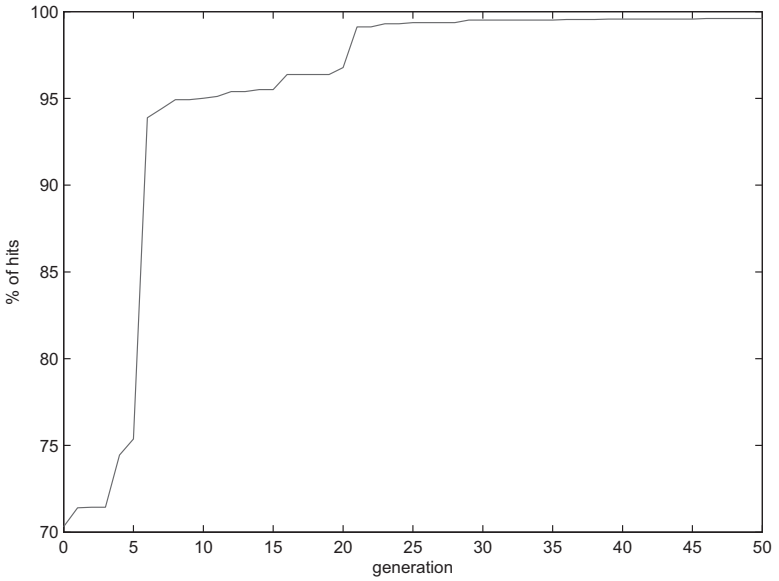

Figure 6: Evolution of the best solution in the Ford run.

6.3 Results

Table 4 shows the results obtained for the Ford classification problem. Due to time constraints the GP algorithm was only run once. As the Ford Classification Challenge included validation series, these results are included as well in the table.

Figure 6 shows the evolution of the run along the generations. It can be seen how the improvement in the solutions slows down while the number of generations increases. In fact, there is not any improvement in the best result in the last 12 generations. In this case, the Wilcoxon signed ranked test was not performed since only one run of the algorithm was executed.

The following equation represents the classifier that obtained the highest percentage of testing hits for the Ford classification problem, that is, the resulting classifying tree.

$$y_n = \ln(f)\left(\left(\frac{x_{n\text{-}3}}{y_{n\text{-}5}} + x_{n\text{-}7} + y_{n\text{-}4}^2\right)\ln(i) + \frac{x_{n\text{-}3}}{y_{n\text{-}5}}\right) \tag{12}$$

$$a = \begin{cases} y_{n\text{-}4}^2 & \text{if } x_{n\text{-}1} \leq \frac{x_{n\text{-}1} - y_{n\text{-}1}}{x_{n\text{-}3}} \\ y_{n\text{-}4} & \text{otherwise} \end{cases}$$

$$b = \begin{cases} x_{n\text{-}3} & \text{if } \ln(x_{n\text{-}3}) \leq y_{n\text{-}5} \\ x_{n\text{-}3}\, y_{n\text{-}7} & \text{otherwise} \end{cases}$$

$$c = \begin{cases} b & \text{if } \frac{y_{n-4}}{x_{n-3}} \leq x_{n-1} \\ a & \text{otherwise} \end{cases}$$

$$d = \begin{cases} y_{n-6} & \text{if } c \ln(x_{n-4}) \leq x_{n-7} - x_n \\ y_{n-1} & \text{otherwise} \end{cases}$$

$$e = \begin{cases} y_{n-6} & \text{if } x_{n-1} \leq \frac{x_{n-1} - y_{n-1}}{y_{n-4}} \\ y_{n-4} & \text{otherwise} \end{cases}$$

$$f = \begin{cases} x_{n-3} + x_{n-8} & \text{if } e \leq y_{n-4} \\ x_{n-5} d & \text{otherwise} \end{cases}$$

$$g = \begin{cases} x_{n-4} - x_{n-3} & \text{if } x_{n-3} \leq y_{n-2} \\ x_{n-5} & \text{otherwise} \end{cases}$$

$$h = \begin{cases} \ln(\ln(x_{n-1} \ln(x_{n-4} - x_{n-6}))) & \text{if } x_{n-7} \leq y_{n-4} \\ \ln(\ln(x_{n-1} \ln(y_{n-5}))) & \text{otherwise} \end{cases}$$

$$i = \begin{cases} \ln(g) \left(\ln(y_{n-8}) + y_{n-4} \left(x_{n-3} + y_{n-4}^2 + \frac{x_{n-3}}{y_{n-5}} h \right) \right) & \text{if } \ln(x_{n-5}) \leq y_{n-4} \\ (y_{n-1} + x_{n-5}) \ln(y_{n-6}) & \text{otherwise} \end{cases}$$

As can be observed, the resulting classifier has a very complex structure that includes nine conditional clauses. It is very difficult to extract any conclusion from Equation (12). In this case, despite obtaining an explicit function as a result, it is equivalent to a black box, given how convoluted its structure is.

The results of the Ford Classification Challenge can be found at http://home.comcast.net/~nn_classification/. Twenty research teams submitted contributions to the competition. The accuracy of the results ranged from a 100% testing hits scored by the first qualified to a 65.9% testing hits scored by the last qualified. We participated in the competition and qualified second. Table 5 shows the results obtained for each of the participant of the Ford competition.

In this case, unfortunately, the organizers of the competition did not publish the classification methods used by the contributors. To the best of our knowledge, the only contributor that published a description of the method used was Paulo Adeodato (Adeodato et al., 2009), which ranked seventh in the competition. The strategy he and his colleagues followed for classifying the data consisted of an initial phase of feature extraction, where more than a hundred features were extracted for each sequence, and a classification phase, where an ensemble of 30 multilayer perceptrons was used. In the paper, the authors emphasize the importance of the feature extraction phase in order to get a good classification. Therefore, they claim that a team working on this kind of problem has to possess expertise on time series, signal processing, or some other kind of sequential data processing.

The proposed GP approach obtained better results in the competition with no need of analyzing the data, preprocessing the data, or performing any feature extraction phase.

Table 5: Results of the Ford Classification Challenge.

Rank	Name	Error (%)	False positive (%)
1	D'yakonov Alexander	100	0
2	Eva Alfaro-Cid	99.6	0
3	Lv Jun	97.8	1.8
4	Cristian Grozea	96.7	2.1
5	Teesid Korsrilabutr	96.5	2.9
6	Schuichi Kurogi	96.0	3.7
7	Paulo Adeodato	95.5	4.6
8	Joerg D. Wichard	95.4	5.0
9	Anthony Bagnall	94.9	3.5
10	Gavin Cawley	94.5	6.6
11	Paavo Nieminen	93.9	6.8
12	David Verstraeten	92.5	7.6
13	Bo Jin	89.6	9.5
14	Dmitry Zhora	85.6	17.2
15	Ukil Abhisek	83.8	15
16	Hugo Jair Escalante	81.6	27.3
17	Paul Chandra	80.7	39
18	Dongrui Wu	76.9	25.8
19	Dymitr Ruta	75.3	40.2
20	Cota Flores Suarez	65.9	24.2

7 Conclusions

This work presents an approach for time series classification that processes time series sequentially, as opposed to a classification based on feature vectors. This approach uses genetic programming and a serial processing of data where the last output of the classifier is considered as the classification result.

The approach was tested in three different problems. Two of the problems are real-world problems proposed for two competitions. The results obtained were compared against those obtained by the participants in the research competition. Our approach provided competitive results for both problems. This shows the potential of the approach.

In the first real world problem tackled in the paper, the classification of EEG signals proposed in the BCI Competition, the GP algorithm converges to a solution that only uses two of the EEG channels recorded in the experiments. These two channels were identified by one of the participants in the competition as the most significant for classification in the time domain. Therefore, GP has been able to automatically identify those data that were more relevant for the problem, avoiding the need for preanalyzing the signals.

In the second real world problem tackled, the classification of automotive signals gathered in the Ford Classification Challenge, the structure of the solution proposed by the GP is quite complex, effectively acting as a black box, despite its explicit function expression.

In both competitions, the participants remarked on the importance of features extraction prior to classification, the difficulty of this process, and the expertise required to perform it. One of the advantages of the method proposed in this work is that the

classification is performed right along with the time series, there is no need of feature extraction or any other preprocessing of data.

The serial processing approach reduces the number of inputs of the system. However, the sliding window effect requires that the evaluation process is applied as many times as there are elements in the sequence. Therefore, one of the drawbacks of serial processing is the time it takes. In effect we are trading off space for time.

Acknowledgments

This work was partially funded by project NoHNES, Non-Hierarchical Network Evolutionary Systems (Spanish Ministerio de Educación y Ciencia, TIN2007-68083-C02).

References

Adeodato, P. J. L., Arnaud, A. L., Vasconcelos, G. C., Cunha, R. C. L. V., Gurgel, T. B., and Monteiro, D. S. M. P. (2009). The role of temporal feature extraction and bagging of MLP neural networks for solving the WCCI 2008 Ford Classification Challenge. In *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2009*, pp. 57–62.

Ahalpara, D. P., and Parikh, J. C. (2006). Modeling time series of real systems using genetic programming. ArXiv Nonlinear Sciences e-prints. Submitted to *Physical Review E.*

Alfaro-Cid, E., Esparcia-Alcázar, A., and Sharman, K. (2006). Using distributed genetic programming to evolve classifiers for a brain computer interface. In M. Verleysen (Ed.), *Proceedings of the European Symposium on Artificial Neural Networks, ESANN-2006*, pp. 59–66.

Alfaro-Cid, E., Merelo, J. J., Fernándezde Vega, F., Esparcia-Alcázar, A. I., and Sharman, K. (2010). Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation*, 18(2): 305–332.

Alfaro-Cid, E., Sharman, K., and Esparcia-Alcázar, A. I. (2006). Evolving a learning machine by genetic programming. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation, CEC-2006*, pp. 958–962.

Arenas, M., Collet, P., Eiben, A., Jelasity, M., Merelo, J., Paechter, B., Preuß, M., and Schoenauer, M. (2002). A framework for distributed evolutionary algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature, PPSN-2002. Lecture notes on Computer Science*, Vol. 2439 (pp. 665–675). Berlin: Springer-Verlag.

Arenas, M., Dolin, B., Merelo, J., Castillo, P., de Viana, I. F., and Schoenauer, M. (2002). Jeo: Java evolving objects. In *Proceedings of the International Conference on Genetic and Evolutionary Computation, GECCO-2002*, pp. 991–994.

Blankertz, B., Müller, K., Curio, G., Vaughan, T., Schalk, G., Wolpaw, J., Schlögl, A., Neuper, C., Pfurtscheller, G., Hinterberger, T., Schröder, M., and Birbaumer, N. (2004). The BCI competition: Progress and perspectives in detection and discrimination of EEG single trials. *IEEE Transactions on Biomedical Engineering*, 51(6): 1044–1051.

Borrelli, A., De Falco, I., Della Cioppa, A., Nicodemi, M., and Trautteura, G. (2006). Performance of genetic programming to extract the trend in noisy data series. *Physica A: Statistical and Theoretical Physics*, 370(1): 104–108.

Dasgupta, D., and Gonzalez, F. (2001). Evolving complex fuzzy classifier rules using a linear tree genetic representation. In *Proceedings of the International Conference on Genetic and Evolutionary Computation, GECCO-2001*, pp. 299–305.

Eads, D., Hill, D., Davis, S., Perkins, S., Ma, J., Porter, R., and Theiler, J. (2002). Genetic algorithms and support vector machines for time series classification. In *Proceedings of the Fifth Conference*

on the Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation. Symposium on Optical Science and Technology of the 2002 SPIE Annual Meeting, pp. 74–85.

Eggermont, J., Eiben, A. E., and van Hemert, J. I. (1999). A comparison of genetic programming variants for data classification. In *Proceedings of the Third International Symposium on Advances in Intelligent Data Analysis, IDA-99. Lecture notes on Computer Science*, Vol. 1642 (pp. 281–290). Berlin: Springer-Verlag.

Esparcia-Alcázar, A. I. (1998). Genetic programming for adaptive signal processing. PhD thesis, Department of Electronics and Electrical Engineering, University of Glasgow.

Estébanez, C., Aler, R., and Valls, J. M. (2007). A method based on genetic programming for improving the quality of datasets in classification problems. *International Journal of Computer Science and Applications*, 4(1): 69–80.

Holladay, K., and Robbins, K. A. (2007). Evolution of signal processing algorithms using vector based genetic programming. In *Proceedings of the IEEE International Conference on Digital Signal Processing*, pp. 503–506.

Hui, A. (2003). Using genetic programming to perform time-series forecasting of stock prices. In J. R. Koza (Ed.), *Genetic Algorithms and Genetic Programming at Stanford 2003* (pp. 83–90). Stanford, CA: Stanford Bookstore.

Jabeen, H., and Baig, A. R. (2010). Review of classification using genetic programming. *International Journal of Engineering Science and Technology*, 2(2): 94–103.

Kaboudan, M. A. (2000). Genetic programming prediction of stock prices. *Computational Economics*, 6(3): 207–236.

Kishore, J. K., Patnaik, L. M., Mani, V., and Agrawal, V. K. (2000). Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation*, 4(3): 242–258.

Kishore, J. K., Patnaik, L. M., Mani, V., and Agrawal, V. K. (2001). Genetic programming based pattern classification with feature space partitioning. *Information Sciences*, 131(1–4): 65–86.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Langdon, W. B., McKay, R. I., and Spector, L. (2010). Genetic programming, 2nd ed. In M. Gendreau and J.-Y. Potvin (Eds.), *Handbook of Metaheuristics*, Vol. 146, *International Series in Operations Research & Management Science*, Chap. 7, pp. 185–225. Berlin: Springer-Verlag.

Liu, Y., and Khoshgoftaar, T. (2004). Reducing overfitting in genetic programming models for software quality classification. In *Proceedings of the Eighth IEEE Symposium on International High Assurance Systems Engineering*, pp. 56–65.

Luke, S. (2000). Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3): 274–283.

Luke, S., and Panait, L. (2006). A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3): 309–344.

Mensh, B., Werfel, J., and Seung, H. (2004). BCI competition 2003, data set Ia: Combining gamma-band power with slow cortical potentials to improve single-trial classification of electroencephalographic signals. *IEEE Transactions on Biomedical Engineering*, 51(6): 1052–1056.

Muni, D. P., Pal, N. R., and Das, J. (2006). Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 36(1): 106–117.

Oltean, M., and Diosan, L. (2009). An autonomous GP-based system for regression and classification problems. *Applied Soft Computing*, 9(1): 49–60.

Otero, F. E. B., Silva, M. M. S., Freitas, A. A., and Nievola, J. C. (2003). Genetic programming for attribute construction in data mining. In *Proceedings of the Sixth European Conference on Genetic Programming, EuroGP-2003. Lecture Notes in Computer Science*, vol. 2610 (pp. 389–393). Berlin: Springer-Verlag.

Poli, R. (2010). Genetic programming theory. In U.-M. O'Reilly (Ed.), *GECCO 2010 Advanced tutorials* (pp. 2473–2502). New York: ACM.

Poli, R., Langdon, W. B., and McPhee, N. F. (2008). *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

Tsakonas, A. (2006). A comparison of classification accuracy of four genetic programming-evolved intelligent structures. *Information Sciences*, 176(6): 691–724.

Wagner, N., and Michalewicz, Z. (2008). An analysis of adaptive windowing for time series forecasting in dynamic environments: Further tests of the DyFor GP model. In *Proceedings of the International Conference on Genetic and Evolutionary Computation, GECCO-2008*, pp. 1657–1664.

Wolpaw, J., Birbaumer, N., Heetderks, W., McFarland, D., Peckham, P., Schalk, G., Donchin, E., Quatrano, L., Robinson, C., and Vaughan, T. (2000). Brain-computer interface technology: A review of the first international meeting. *IEEE Transactions on Rehabilitation Engineering*, 8(2): 164–173.