GenClass: A parallel tool for data classification based on Grammatical Evolution

Nikolaos Anastasopoulos(1), Ioannis G. Tsoulos(2), Alexandros Tzallas(2)

(1) *Department of Electrical and Computer Engineering, University of Patras, Greece*
(2) *Department of Informatics and Telecommunications, University of Ioannina, 47100 Arta, Greece*

Abstract

A genetic programming tool is proposed here for data classification. The tool is based on Grammatical Evolution technique and its designed to exploit multicore computing systems using the OpenMp library. The tool constructs classification programs in a C – like programming language in order to classify the input data, producing simple if – else rules and upon termination the tool produces the classification rules in C and Python files. The tool is tested on a wide range of classification problems and the produced results are compared against traditional techniques for data classification, yielding very promising results.

Keywords: Genetic algorithm, Data classification, Grammatical evolution, Stochastic methods

1. Introduction

Data classification finds many applications on a series of practical problems from areas such as chemistry [1, 2, 3], biology [4, 5], economics [6, 7], physics [8, 9] etc. During the past years many methods have been proposed to problems of this category such as neural networks [10], radial basis functions networks [11], support vector machines [12], etc. The proposed method, which is initially described in [13], constructs classification programs in a human readable format using the technique of Grammatical Evolution [14]. Grammatical evolution is an evolutionary process that has been applied with success in many areas such as music composition [15], economics [16], symbolic regression [17], robot control [18] and caching algorithms [23].

The rest of this article is organized as follows: in section 2 the software is described in detail, in section 3 a series of experiments on some well - known classification datasets are demonstrated and finally in section 4 conclusions and guidelines for further expansion of the software are presented.

2. Software description

2.1. The proposed algorithm

The main steps of the algorithm are:

1. **Initialization** step.
   (a) Read the train data
   (b) Set N_G the maximum number of generations, N_C as the number of chromosomes, P_S the selection rate, P_M the mutation rate.
   (c) Initialize the chromosomes of the population.
2. **Genetic** step
   (a) **For** $i = 1, \ldots, N_g$ **do**
      i. Create for every chromosome in the population a classification program using Grammatical Evolution.
      ii. Calculate the fitness for every chromosome of the population
      iii. Execute the genetic operators of selection with rate P_S and mutation with rate P
   (b) **EndFor**
3. **Evaluation** step
   (a) Create a classification program for the best chromosome in the population.
   (b) Apply the previous program to test set and report the induced error.

2.2. Installation

The package is distributed in a tar.gz file named `GenClass.tar.gz` and under UNIX systems the user must issue the following commands to extract the associated files:

1. gunzip `GenClass.tar.gz`
2. tar xfv `GenClass.tar`
3. `cd GenClass`
4. Edit the file `Makefile.inc` and change (if needed) the configuration parameters.
5. Type `make`.

The parameters in `Makefile.inc` are the following:

1. **CXX**: It is the most important parameter. It specifies the name of the C++ compiler. In most systems running the GNU C++ compiler this parameter must be set to g++.
2. **ROOTDIR**: Is the location of the GenClass directory.

2.3. The executable genclass

The outcome of the compilation is the executable `genclass` under the directory `bin`. The executable has the following command line parameters:

1. `-h`:The program prints a help screen.
2. `-c count`: The parameter `count` determines the number of chromosomes with default value 500.
3. -f **count**. Specify fold count for fold validation. Default value 0 (no folding).
4. -g **gens**: The parameter **gens** determines the maximum number of generations with default value is 200.
5. -d **count**. The parameter d determines the maximum number of threads used by the OpenMp library. The default value is 16.
6. `-s srate`: The parameter `srate` specifies the selection rate with default value 0.10 (10%).
7. `-m mrate`: The parameter `mrate` specifies the mutation rate with default value 0.05 (5%).
8. -l **size**: The parameter **size** determines the size of every chromosome with default value 100.
9. -p **train_file**: The string parameter **train_file** specifies the file containing the points that will be used as train data for the algorithm. The file should conform to the format outlined in figure 1. The integer value D determines the dimensionality of the problem and the value M determines the number of points in the file. Every subsequent line contains a pattern and the final column is the real output (category) for this pattern.
10. -t **test_file**: The string parameter **test_file** specifies the file containing the test data for the particular problem. The file should be in the same format as the **train_file**.
11. -o **method**: The string parameter method specifies the output method for the executable. The available options are
    (a) simple. The program prints output only on termination.
    (b) csv. In every generation the program prints: number of generations, train error and test error. This is the default value for the string parameter **method**.
    (c) full. The program prints in every generation detailed information about the optimization procedure as well as classification error for every distinct class of the problem.
12. `-r seed`: The integer parameter `seed` specifies the seed for the random number generator. It can assume any integer value.

2.4. The output files

The software produces upon termination two distinct files that contains the classification rules. The first file is written in ANSI C++ named classifier.c and an example is shown in Figure 2. The second file is written in Python named classifier.py and an example is displayed in Figure 3.

3. Experiments

3.1. A typical example

Consider the Ionosphere dataset available from the Machine Learning Repository. The ionosphere dataset contains data from the Johns Hopkins Ionosphere database. The dataset has been divided into two files, `ionosphere.train` and `ionosphere.test` under directory `examples` of the distribution. A typical run for the `GenClass` will be

```
../bin/genclass -p ionosphere.train -t ionosphere.test -g 10 -o csv
```

The output of this command is shown in Figure 4.

3.2. Experiments

In order to measure the efficiency of the proposed method a series of experiments were conducted on some common classification problems found in two major dataset databases:

1. UCI dataset repository, `https://archive.ics.uci.edu/ml/index.php`
2. Keel repository, `https://sci2s.ugr.es/keel/datasets.php`[19].

All the experiments were conducted 30 times using different seed for the random generator each time and averages were taken. In all experiments we have used the parameters shown in table 1. The following datasets were used

1. **Wine** dataset. The wine recognition dataset contains data from wine chemical analysis.
2. **Glass** dataset. The dataset contains glass component analysis for glass pieces that belong to 6 classes.
3. **Tae** dataset. The data consist of evaluations of teaching performance for the University of Wisconsin-Madison.
4. **Spiral** dataset: The spiral artificial dataset contains 1000 two-dimensional examples that belong to two classes (500 examples each). The number of the features is 2. The data in the first class are created using the following formula: $x_1 = 0.5t\cos(0.08t)$, $x_2 = 0.5t\cos\left(0.08t + \frac{\pi}{2}\right)$ and the second class data using: $x_1 = 0.5t\cos(0.08t + \pi)$, $x_2 = 0.5t\cos\left(0.08t + \frac{3\pi}{2}\right)$

5. **Pima** dataset. The Pima Indians Diabetes dataset contains 768 examples of 8 attributes each that are classified into two categories: healthy and diabetic.
6. **Ionosphere** dataset. The ionosphere dataset (ION in the following tables) contains data from the Johns Hopkins Ionosphere database.
7. **Appendictis** dataset, proposed in [20].
8. **Australian** dataset, the dataset concerns credit card applications.
9. **Hayes roth** dataset. This dataset[21] contains **5** numeric-valued attributes and 132 patterns.
10. **Alcohol** dataset, a dataset about Alcohol consumption [22].
11. **Dermatology** dataset. Dataset used for differential diagnosis of erythemato-squamous diseases.
12. **Balance** dataset. This data set was generated to model psychological experimental results.
13. **Regions2** dataset. It is created from liver biopsy images of patients with hepatitis C [25].
14. **Parkinsons** dataset. This dataset is composed of a range of biomedical voice measurements from 31 people, 23 with Parkinson's disease (PD)[24].
15. **Wdbc** dataset. The Wisconsin diagnostic breast cancer dataset (WDBC) contains data for breast tumors.
16. **Popfailures** dataset. This dataset contains records of simulation crashes encountered during climate model uncertainty quantification (UQ) ensembles.
17. **Heart** dataset. The task is to detect the absence or presence of heart disease.
18. **Ecoli** dataset. The goal here is to predict the localization site of proteins.
19. **Haberman** dataset. A dataset about breast cancer from a study at the University of Chicago's Billings Hospital.
20. **HouseVotes** dataset. This data set includes votes for each of the U.S. House of Representatives Congressmen on the 16 key votes.
21. **Shuttle** dataset. The task is to decide what type of control of a spacecraft should be employed.
22. **Lymography** dataset. The aim here is to detect the presence of a lymphoma in patients.
23. **Mammographic** dataset. This dataset be used to identify the severity (benign or malignant) of a mammographic mass lesion from BI-RADS attributes and the patient's age.

24. **OptDigits** dataset. Optical Recognition of Handwritten Digits data set.
25. **Page Blocks** dataset. The dataset contains blocks of the page layout of a document that has been detected by a segmentation process.
26. **Penbased** dataset. This is a Pen-Based Recognition of Handwritten Digits data set with 10992 patterns of 16 features.
27. **Saheart** dataset. The dataset is about to categorize persons if have a coronary heart disease.
28. **Segment** dataset. This database contains patterns from a database of 7 outdoor images (classes).
29. **Thyroid** dataset. The goal in this dataset is to detect if a patient is normal or suffers from hyperthyroidism or hypothyroidism.
30. **Eeg** dataset. As an real word example, consider an EEG dataset described in [26] is used here. The dataset consists of five sets (denoted as Z, O, N, F and S) each containing 100 single-channel EEG segments each having 23.6 sec duration. With different combinations of these sets the produced datasets are Z_F_S, ZO_NF_S, ZONF_S and Z_O_N_F_S.

The results from the experiments are displayed in table 2. It is evident, that the proposed method outperforms the other methods in terms of efficiency in the majority of the objective problems. Of course, the method could be slower than RBF or Neural network due to the usage of Grammatical Evolution, however this increase in time can be significantly reduced with the use of threads.

4. Conclusions

A software which implements a method for data classifications was introduced. The method is based on grammatical evolution and the software is designed to be portable. The software is entirely written in ANSI C++ and there is not any specific software requirement. Future versions of the software will include

- Input from various formats such as CSV, Json etc.
- Usage of improvement stopping rules for the genetic algorithm.
- Additional output formats.

References

[1] C. Güler, G. D. Thyne, J. E. McCray, K.A. Turner, Evaluation of graphical and multivariate statistical methods for classification of water chemistry data, Hydrogeology Journal **10**, pp. 455-474, 2002

[2] E. Byvatov ,U. Fechner ,J. Sadowski , G. Schneider, Comparison of Support Vector Machine and Artificial Neural Network Systems for Drug/Nondrug Classification, J. Chem. Inf. Comput. Sci. **43**, pp 1882–1889, 2003.

[3] Kunwar P. Singh, Ankita Basant, Amrita Malik, Gunja Jain, Artificial neural network modeling of the river water quality—A case study, Ecological Modelling **220**, pp. 888-895, 2009.

[4] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene Selection for Cancer Classification using Support Vector Machines, Machine Learning **46**, pp. 389-422, 2002.

[5] R. Marabini, J.M. Carazo, Pattern recognition and classification of images of biological macromolecules using artificial neural networks, Biophysical Journal **66**, pp. 1804-1814, 1994.

[6] I. Kaastra, M. Boyd, Designing a neural network for forecasting financial and economic time series, Neurocomputing **10**, pp. 215-236, 1996.

[7] Moshe Leshno, Yishay Spector, Neural network prediction analysis: The bankruptcy case, Neurocomputing **10**, pp. 125-147, 1996.

[8] S. R. Folkes,O. Lahav, S. J. Maddox, An artificial neural network approach to the classification of galaxy spectra, Montly Notices of the Royal Astronomical Society **283**, pp 651-665, 1996.

[9] C.Z. Cai, W.L. Wang, Y.Z. Chen, Support vector machine classification of physical and biological datasets, Int. J. Mod. Phys. C **14**, pp. 575, 2003

[10] K. Hornik, Multilayer feedforward networks are universal approximators, Neural Networks **2**, pp. 359-366, 1989

[11] M.D. Buhmann, Radial basis functions, Cambridge monographs on Applied and Computational Mathemetics, 2003.

[12] I. Steinwart, A. Christmann, Support Vector Machines, Information Science and Statistics, Springer, 2008.

[13] Ioannis G. Tsoulos, Creating classification rules using grammatical evolution, International Journal of Computational Intelligence Studies **9**, pp. 161-171, 2020.

[14] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Trans. Evol. Comput. **5**, pp. 349–358, 2001.

[15] Alfonso Ortega, Rafael Sánchez, Manuel Alfonseca Moreno, Automatic composition of music by means of grammatical evolution, APL '02 Proceedings of the 2002 conference on APL: array processing languages: lore, problems, and applications Pages 148 - 155.

[16] Michael O'Neill, Anthony Brabazon, Conor Ryan, J. J. Collins, Evolving Market Index Trading Rules Using Grammatical Evolution, Applications of Evolutionary Computing Volume 2037 of the series Lecture Notes in Computer Science pp 343-352.

[17] M. O'Neill, C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitary Language, Genetic Programming, vol. 4, Kluwer Academic Publishers, Dordrecht, 2003

[18] J.J. Collins, C. Ryan, in: Proceedings of AROB 2000, Fifth International Symposium on Artificial Life and Robotics, 2000.

[19] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera. KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework. Journal of Multiple-Valued Logic and Soft Computing 17, pp. 255-287, 2011.

[20] Weiss, Sholom M. and Kulikowski, Casimir A., Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems, Morgan Kaufmann Publishers Inc, 1991.

[21] B. Hayes-Roth, B., F. Hayes-Roth. Concept learning and the recognition and classification of exemplars. Journal of Verbal Learning and Verbal Behavior **16**, pp. 321-338, 1977.

[22] Tzimourta, Katerina D. and Tsoulos, Ioannis and Bilero, Thanasis and Tzallas, Alexandros T. and Tsipouras, Markos G. and Giannakeas, Nikolaos, Direct Assessment of Alcohol Consumption in Mental State Using Brain Computer Interfaces and Grammatical Evolution, Inventions **3**, pp. 1-12, 2018.

[23] M. O'Neill, C. Ryan, in: K. Miettinen, M.M. Mkel, P. Neittaanmki, J. Periaux (Eds.), Evolutionary Algorithms in Engineering and Computer Science, Jyvskyl, Finland, 1999, pp. 127–134.

Figure 1: Data format.

D

M

$$\begin{array}{ccccc} x_{11} & x_{12} & \dots & x_{1D} & y_1 \\ x_{21} & x_{22} & \dots & x_{2D} & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{M1} & x_{M2} & \dots & x_{MD} & y_M \end{array}$$

Figure 2: A typical produced C file of the tool.

```
#include <math.h>
int classifier(double *x)
{
        int CLASS = 0;
        if(!(x[6]<x[25]||x[4]<(((x[5]-((-5.79)/x[0]))/
            (((-503.562)/x[22])/(x[20]/x[24])))/(x[0]+(x[23]/x[4])))
            ||!(x[23]<=x[2]))) CLASS=0;
        else CLASS=1;
        return CLASS;
}
```

[24] Little MA, McSharry PE, Hunter EJ, Spielman J, Ramig LO. Suitability of dysphonia measurements for telemonitoring of Parkinson's disease. IEEE Trans Biomed Eng. 2009;56(4):1015. doi:10.1109/TBME.2008.2005954

[25] Giannakeas, N., Tsipouras, M.G., Tzallas, A.T., Kyriakidi, K., Tsianou, Z.E., Manousou, P., Hall, A., Karvounis, E.C., Tsianos, V., Tsianos, E. A clustering based method for collagen proportional area extraction in liver biopsy images (2015) Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS, 2015-November, art. no. 7319047, pp. 3097-3100.

[26] R.G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger, Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state, Phys. Rev. E **64**, pp. 1-8, 2001.

Figure 3: A typical output file in Python language.

```
import ctypes
import numpy as np from typing
import List
def classifier(input: List):
    fun = ctypes.CDLL("./classifier.so")
    fun.classifier.argtypes = [ctypes.POINTER(ctypes.c_double)]
    fun.classifier.restype = ctypes.c_int
    a = np.array(input)
    input_ptr = a.ctypes.data_as(ctypes.POINTER(ctypes.c_double))
    return fun.classifier(input_ptr)
```

Figure 4: Typical output of the GenClass command

```
 1,      15.43,      19.32
  2,      15.43,      19.32
  3,      15.43,      19.32
  4,      13.71,      17.05
  5,      12.57,      15.34
  6,      12.57,      15.34
  7,      12.57,      15.34
  8,         12,      13.64
  9,         12,      13.64
  FINAL OUTPUT
  EXPRESSION=
  if(!(x7<log(cos(cos(((-788.787)+((sin(x28)/sin(cos(((-7.17)/x34))))
      +(-83.6))))))|x6>x13&x7<log(x5)))
  CLASS=0.00   else   CLASS=1.00
TRAIN ERROR = 12.00%
 CLASS ERROR = 13.64%
```

Table 1: Parameters for the experiments.

PARAMETER	VALUE
N_C	500
N_G	500
P_S	90%
P_M	5%

Table 2: Results.

DATASET	NEURAL	RBF	GENCLASS
Alcohol	26.85%	46.63%	14.68%
Appendicitis	22.50%	12.23%	15.00%
Australian	30.48%	34.89%	14.57%%
Balance	8.19%	33.42%	0.00%
Dermatology	14.33%	62.34%	3.72%
Ecoli	53.60%	59.59%	24.54%
Glass	54.24%	50.16%	33.81%
Haberman	29.94%	25.10%	27.33%
Hayes Roth	35.16%	64.36%	25.39%
Heart	25.95%	31.20%	17.55%
HouseVotes	7.54%	6.13%	3.72%
Ionosphere	15.83%	16.22%	8.29%
Liverdisorder	33.82%	30.84%	32.06%
Lymography	25.57%	25.31%	19.29%
Mammographic	27.08%	21.38%	17.35%
OptDigits	50.37%	81.22%	24.33%
Page Blocks	7.10%	10.09%	3.82%
Parkinsons	19.44%	17.42%	9.47%
Penbased	42.91%	82.47%	25.32%
Pima	29.07%	25.78%	25.59%
Popfailures	6.57%	7.04%	7.04%
Regions2	33.32%	38.29%	19.52%
Saheart	33.22%	32.19%	31.30%
Segment	24.54%	59.68%	10.52%
Shuttle	31.29%	32.97%	0.15%
Spiral	42.60%	44.87%	36.90%
Tae	47.53%	60.07%	37.33%
Thyroid	4.28%	10.52%	1.89%
Wdbc	20.55%	7.27%	4.11%
Wine	46.63%	31.41%	4.71%
Z_F_S	14.17%	13.16%	7.00%
Z_O_N_F_S	78.73%	48.71%	32.20%
ZO_NF_S	9.97%	9.02%	2.60%
ZONF_S	3.48%	4.03%	1.60%

Required Metadata

Current code version

Nr.	**Code metadata description**	
C1	Current code version	1.0
C2	Permanent link to code/repository used for this code version	`https://github.com/itsoulos/GenClass/`
C3	Legal Code License	GNU General Public License (GPL)
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	C++
C6	Compilation requirements, operating environments & dependencies	Linux
C7	If available Link to developer documentation/manual	`https://github.com/itsoulos/GenClass/wiki`
C8	Support email for questions	itsoulos@uoi.gr