Extracting Knowledge from Databases with Genetic Programming: Iris Flower Classification Problem

Daniel Rivero[1], Juan R. Rabuñal[2], Julián Dorado[2], Alejandro Pazos[2] and Nieves Pedreira[2]

Univ. da Coruña, Fac. Informática, Campus Elviña, 15192 A Coruña, Spain
[1] infdrc00@ucv.udc.es [2] {juanra, julian, ciapazos, nieves}@udc.es

Abstract. In the world of databases the extraction of knowledge has been a very useful tool for many different purposes and tried with many different techniques. In this paper we use Genetic Programming (GP) to solve a classification problem from a database and we will show how we can adapt this tool in two different ways: to improve its performance and to make possible the detection of errors. Results show that the technique developed in this paper opens a new area for research in the field extracting knowledge from more complicated structures, like neural networks.

1 Introduction

Genetic Programming (GP) [1] is an evolutionary method that creates computer programs that represent approximate or exact solutions to a problem. This technique allows the finding of programs with the shape of a tree, and in its most common application those programs will be mathematical expressions combining mathematical operators, input variables, constants, decision rules, relational operators, etc.

All of these possible operators must be specified before starting the search, and so with them GP must be able to build trees with the objective of finding the desired expression which models the relation between the input variables and the desired output. This set of operators are divided into two groups: terminal set, with the ones which can not accept parameters, like variables or constants; and function set, with the ones which need parameters, like add or subtract operators. Once the terminal and non-terminal operators are specified, it is possible to establish types: each node will have a type, and the construction of child expressions needs to follow the rules of the nodal type [2].

GP makes a process of automatic program generation by means of a process based on Darwin's evolution theory [3], in which, after subsequent generations, new trees (individuals) are produced from old ones by means of crossover, copy and mutation [4] [5], based on natural selection: the best trees will have more chances of being chosen to become part of the next generation. Thus, a stochastic process is established in which, after successive generations, obtains a well adapted tree.

As the programs we are obtaining with GP have the shape of trees, GP has the ability of adapting to many different kinds of problems. The problem proposed in this paper is extracting knowledge from databases, and we will show how we can solve it with GP in two different ways in a classification problem: by extracting a rule (with the shape of an IF-THEN-ELSE rule) that makes classifications, and extracting different rules, each one for each classification class.

In the field of knowledge discovery from databases one of the most successful applications of GP is in the development of fuzzy rules [6] [7], mixing its ability to develop rules, and using the technique of Automatically Defined Functions (ADF), described in [8], for obtaining fuzzy rules.

In a recent work done by Wong and Leung GP is applied as a knowledge extraction technique from databases, and they present LOGENPRO (Logic Grammar Based Genetic Algorithm) [9]. They make a combination of GP and representation of knowledge in first order logic. This first approximation shows the advantages of GP as a KDD (Knowledge Discovery in Databases) extraction technique.

GP was also used as a rule extraction technique in combination with decision trees, where the functions in the nodes of the trees use one or more variables [10], but this combination makes the algorithm design very complicated. More recently, Engelbrecht, Rouwhorst and Schoeman [11] apply GP and decision trees for extracting knowledge from databases designing an algorithm called BGP (Building-Block Approach to Genetic Programming). In this algorithm GP is combined with decision trees, but, in this case, centered in the concept of building block, which represents a condition or a node of the tree. A building block has three parts: an attribute, a relational operator and a threshold. Rules are obtained by combining different values of the parts of the building blocks in the shape of decision trees.

2 Description of the problem

The iris flower data [12] were originally published by Fisher [13] for examples in discriminant analysis and cluster analysis. Four parameters, including *sepal length, sepal width, petal length* and *petal width*, were measured in millimeters on fifty iris specimens from each of three species, *Iris setosa*, *Iris versicolor*, and *Iris virginica*. So, given the four parameters, one should be able to determine which of the three classes a specimen is categorized to. There are 150 data points listed in the database.

One of the reasons for applying this problem is due to the physical situation of the classes in the four-dimension space. On Fig 1. can be seen the space distribution for variables X_1 and X_2 (petal length and petal width). As shown on [14], with these two variables we can get a higher discrimination for the three classes, a fitness of a 98% of success using only these two variables. So, they are an important reference point for comparing graphically the results.

In this paper we will show how we can use GP to solve the iris flower problem. We will see two different points of view. In the first, we will use GP in order to obtain a rule classifier system (one-tree classification), and in the second we will try to find a boolean expression for each of the three species to determine if the data belongs to that class (three-tree classification). We will see how GP seems to be a suitable technique not just for classify problems, but in general also for extracting knowledge from databases and data mining.

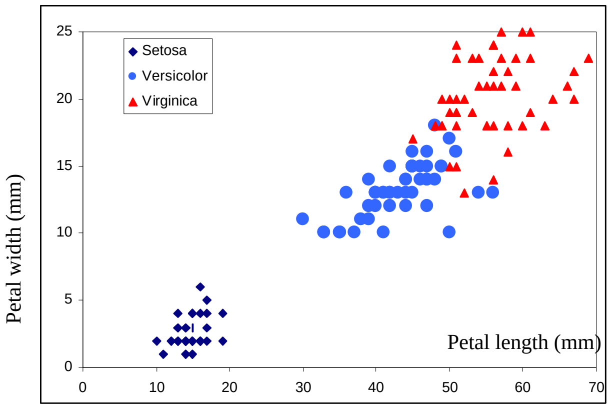

Fig. 1. Distribution of the three classes.

3 One-tree classification

In this part we will configure and run GP in order to obtain a single tree that makes a classification of the data points. Here we will show how we can improve the performance of GP by pre-processing the data and this way obtain better results.

3.1 Classification with no pre-processing

Here we will solve the problem with the data taken as is: with no modification at all.

3.1.1 Configuration

As explained in section 1, to make possible the run of GP, we need to specify the terminal and function sets.

As we want to obtain a flower classification, we will need to make trees with a concrete structure: we will use the typing properties of GP to do this. We will ask GP to make trees with a special type: FLOWER_TYPE.

To have the trees as classifier rules, we just have three terminals and one function returning that type. These terminals are *Setosa*, *Virginica* and *Versicolor*, one for each type of flower. The function is IF-THEN-ELSE, which accepts as first input a boolean expression and as second and third inputs expressions with FLOWER_TYPE type, whether they are one of the three terminals or other IF-THEN-ELSE expressions.

So, the resulting trees will have the shape of a decision rule, for example:

```
IF <boolean expression>
THEN
  IF <boolean expression>
  THEN **Virginica**
  ELSE **Versicolor**
ELSE
  IF <boolean expression>
  THEN
  ELSE **Versicolor**
```

To build the boolean expressions, we will use as terminals the variables X_1, X_2, X_3 and X_4, which stand for *petal width*, *petal length*, *sepal width* and *sepal length* respectively (all four variables real numbers); and the random constants, extracted from the real interval between 1 and 80. We chose this interval because it contains the maximum and minimum values those four variables can have.

We also have the traditional arithmetic operators +, -, * and %, standing % for the division protected operator, which returns a value of 1 if the denominator is equal to zero.

For building boolean expressions, relational and boolean operands are required; relational for establishing relations between the real expressions (containing those four variables and constants) and boolean operators for joining other boolean expressions if necessary.

The complete set of terminals and functions, with their types and their children types, in case of being functions, can be seen on table 1.

	Name	**Returning type**	**Parameter type**
Terminal set	X_1, X_2, X_3, X_4	REAL	
	[1, 80]	REAL	
	Setosa, Verginica, Versicolor	FLOWER_TYPE	
Function set	IF-THEN-ELSE	FLOWER_TYPE	BOOLEAN, FLOWER_TYPE, FLOWER_TYPE
	+, -, *, %	REAL	REAL, REAL
	<, >,>=,<=	BOOLEAN	REAL, REAL
	AND, OR	BOOLEAN	BOOLEAN, BOOLEAN
	NOT	BOOLEAN	BOOLEAN

Table 1. Terminal and function sets used for the one-tree classification.

3.1.2 Results

For solving this problem different combinations of parameters were used, and the set with which we obtained better results can be seen on table 2.

Selection algorithm	Tournament
Crossover rate	95%
Mutation rate	4%
Population size	500 individuals

Parsimony level	0.0001

Table 2. Best parameters for solving with GP.

The result is a classification rule with a fitness of 99.33% success: it fails one case out of the 150 possible. The rule obtained, after 23 hours of computing in an AMD K7 at 1Ghz and 128 MB of RAM memory, is the following:

```
IF ($X_2$ < 25.370)
THEN Setosa
ELSE
  IF (((($X_2$-40.959)%$X_2$)<($X_2$-($X_3$-($X_1$%($X_1$-($X_4$-23.969)))) AND
     (($X_2$%($X_2$-42.760)<($X_2$-(23.969%($X_2$-($X_4$-($X_2$-34.507))))))
     AND ((($X_1$-10.856)%($X_2$-40.959))<($X_2$-($X_3$-(($X_2$-42.760)%
     ($X_1$-21.777))))))) AND ((($X_2$-$X_3$ < ($X_4$-($X_2$%($X_2$-($X_4$-($X_2$-
     ($X_4$-($X_2$-($X_3$-(($X_2$-40.959)%($X_1$-21.777))))))))))))   AND
     ((($X_2$ % ($X_2$-($X_4$-($X_2$-($X_3$-($X_1$ % ($X_2$-40.959))))))) % ($X_2$-
     40.959))<$X_2$-($X_3$-($X_1$%($X_1$-21.777))))) AND ((23.969%($X_1$
     -10.856))<(($X_2$-(23.969%($X_2$-($X_4$-($X_2$-($X_3$-(($X_2$-40.959) %
     ($X_2$-$X_3$-21.777))))))))-($X_2$%($X_2$-($X_4$-($X_2$-40.959))))))))
     AND (($X_3$-(($X_2$-40.959)%($X_1$-21.777)))>21.777))
   THEN Virginica
   ELSE Versicolor
```

3.2 Classification with pre-processing

Now we will see how we can improve the performance of GP with a pre-processing of the data. What we will do is to normalize the data to make all four parameters be in the interval [0,1]. We do this to make these four variables be in the same rank and so for GP it will be easier to make and combine mathematical expression including constants because these will be in similar ranks, and so we won´t be combining expressions with out-of-rank values.

3.2.1 Configuration

As we still want to obtain one tree with the shape of an IF-THEN-ELSE classifier rule, we will use a terminal and function set similar to the ones explained in section 3.1.1 and shown in table 1. The only exception is the use of random constants: now they will not be within the rank [1,80], but in the rank [0,1] because as we have the data normalized in that rank, we need the constants to be in the same rank too.

3.2.2 Results

The best set of parameters found for solving this problem is the same as described in section 3.1.2, shown in table 2. With these parameters, the best expression found, with a fitness of 100%, computed with the same machine and after 7 hours, is the following:

```
IF ($X_2$ < 0.373)
THEN Setosa
ELSE
   IF  (((0.483>(($X_4$-0.799)*(0.483%($X_1$-$X_2$))))  AND  (($X_3$>
     0.701) OR ($X_4$>($X_1$-(0.182+X3)*(0.483%($X_1$-$X_2$)))))) OR
     (((-0.132)*(($X_1$*$X_2$)%((0.821*(0.721-($X_1$-(0.182+$X_3$))))
     -(0.182+$X_3$))))>(0.483%($X_1$-0.701))))
   THEN
     IF (((($X_4$>(-0.132*(($X_1$*$X_1$)%(0.721-(0.182+$X_3$)))) AND
       ($X_4$>((-0.132) * (0.483%(0.721-((($X_1$-((0.182+$X_3$) *
       0.877))*(0.483%($X_1$-$X_2$)))*(0.483%(0.182+$X_3$))))))))
       AND  (0.821>((-0.132)*(0.483%($X_1$-0.701)))))  AND
       ($X_4$>(-0.132*($X_4$ % (0.721-((($X_1$-(0.182+$X_3$))*(0.483%
       ($X_1$-$X_2$)))*(0.483%(0.182+$X_3$)))))))) OR (($X_3$>($X_1$-(-
       0.132))) OR (($X_2$>0.721) AND ($X_4$>((-0.132)*(0.483%
       ($X_1$-((0.182+$X_3$)*0.877))))))))
     THEN
       IF (((0.721>($X_2$*$X_4$)) AND ($X_4$>$X_2$)) OR ((($X_1$<0.877)
         OR (0.799>$X_4$)) OR ($X_3$>0.799)))
       THEN Verginica
       ELSE Versicolor
     ELSE Versicolor
   ELSE Versicolor
```

Note that the second expression (obtained with pre-processing) has better fitness, and it has been found in less time, so with a small pre-processing of the data we improved the performance of GP, in the fitness obtained and in the time taken to develop the desired expression.

4 Three-tree classification

In this section we will solve the problem in a different point of view. Now we will use GP not for making a simple classification into one of the three classes, but to extract three boolean rules to determine whether a particular data point belongs to each specie of flower.

As we have three decision rules, it will have an additional advantage: if, for the same data point, none of the rules make an output as true, or if more than one make an output as true, then we can conclude that this point is now well classified by the system, i.e., we can detect some errors made by the system.

4.1 Configuration

To obtain these boolean rules, we configure GP for obtaining boolean expressions: now the resulting type will be BOOLEAN. As boolean elements we will have the IF-THEN-ELSE classifier rules (accepting three children with BOOLEAN type), relational operators, needed for establishing relations between variables and constants, and boolean operators.

We will also need the four variables and random constants, now inside the interval [0,1], because now we are working directly with normalized values. The complete terminal and function set is shown on table 3.

	Name	Returning type	Parameter type
Terminal set	X_1, X_2, X_3, X_4	REAL	
	[0, 1]	REAL	
Function set	IF-THEN-ELSE	BOOLEAN	BOOLEAN, BOOLEAN, BOOLEAN
	+, -, *, %	REAL	REAL, REAL
	<, >,>=,<=	BOOLEAN	REAL, REAL
	AND, OR	BOOLEAN	BOOLEAN, BOOLEAN
	NOT	BOOLEAN	BOOLEAN

Table 3. Terminal and function sets used for the three-tree classification

4.2 Results

The set of parameters which gave better results are the same as described in section 3.1.2, and shown on table 2. With these elements, the results obtained, with the inputs already normalized, can be seen on table 4. Note that the third expression (used for *Iris Virginica*) has a fitness of 99.33% success, that is, it fails on one case out of the 150. This case is classified by the system as both Virginica and Versicolor, which is an invalid exit and it is detected. So we can consider that the system doesn´t have any fail and gets a fitness of 100% success.

Flower type	Expression obtained	Fitness
Setosa	(X_1 < 0.3141)	100%
Versicolor	(((0.677>X_3) OR (0.526<X_2<(0.736))) AND (((0.610<X_1<0.721) OR ((0.3360<X_1<0.526) OR (0.526<X_2< 0.721))) AND ((X_3>X_1) OR (0.677>X_1))))	100%
Virginica	(((X_1>X_2) OR (X_2>0.718)) AND ((X_2>X_4) OR (((0.739<X_2<0.765) OR (X_4>0.902)) OR (X_1>X_3))))	99.33%

Table 4. Expressions obtained for classifying into the three different classes.

The comparison with other techniques can be seen on the following table:

Method	Type	Fitness	Reference
Proposed here	Rules	100%	
ReFuNN	Fuzzy	95.7%	[15]
C-MLP2LN	Crisp	98.0%	[14]
SSV	Crisp	98.0%	[14]
ANN	Weigths	98.67%	[16]
Grobian	Rough	100.0%	[17]
GA+NN	Weigths	100.0%	[18]
NEFCLASS	Fuzzy	96.7%	[19]
FuNe-I	Fuzzy	96.0%	[20]

Table 5. Comparison between the method proposed here and other different methods.

The distributions obtained from these three rules can be seen on Fig. 2. In this graph, as in the following, the X axis references petal width (X_1) and Y axis references petal length (X_2).

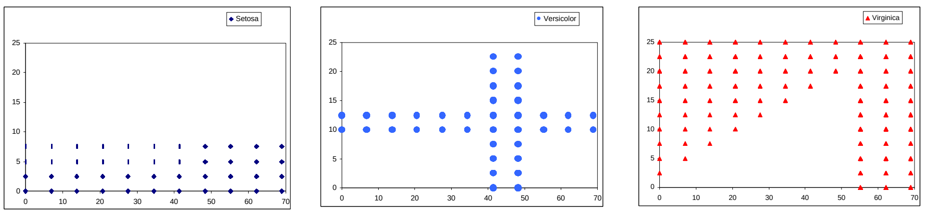

Fig. 2. Distributions obtained for the three classes

We can put these three distributions together in the same graph and compare them with the training set. This is shown on Fig 3.

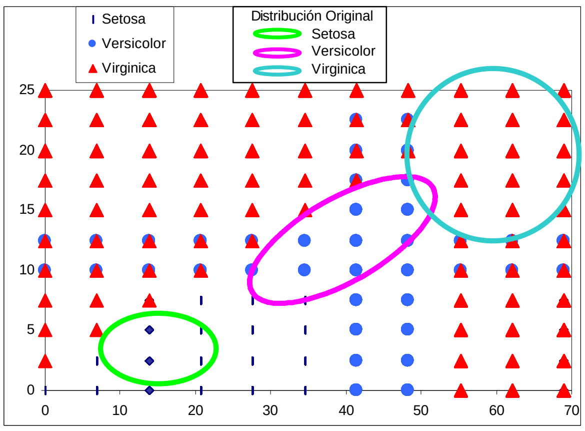

Fig. 3. Distributions obtained from the rules and from the training set.

In this figure we can see that the rule extraction system tries to join those values which depend on each classification and tries to isolate them from those values dependent on other classifications. The intersection areas are those in which the system makes incorrect outputs indicating that the given output is not correct and an individual analysis is necessary for those values to determine which class they belong.

5 Conclusions

As shown in the results, GP seems to be a powerful technique for extracting knowledge from databases. In this paper it has been applied to a well known problem, the iris flower data, with good results and the additional advantage of having as results mathematical expressions which relations the parameters.

In the first attempt, the one-tree classifier, we show how we can adapt GP to produce decision rules with the desired shape, and so how we can obtain high-level explicit knowledge about the system. In this part we can also see that it is better to do a pre-processing to the data to improve the performance of GP. This is so because we put all of the parameters to the same rank and so the system finds easier to work with all variables and constants in the same rank (i.e., it does not find problems in combining constants and values with much different values). We can conclude that with a minimum analysis of the data we can improve the process in both ways: in the final success and in the time needed to obtain it.

The second attempt, the three-tree classifier, gave additional knowledge. With the construction of three different boolean expressions, each one for each class, we obtained an additional knowledge: now we can detect errors made by the system.

GP, then, is shown to be a suitable technique for extracting knowledge from databases, not only in classification problems: its ability to adapt to many different environments (the user selects which operator is needed to be included in the sets) allows the extraction of mathematical relations, decision rules, etc.

6 Future works

As we obtained better results with normalized parameters and different classification rules, the following step is to try to extract knowledge from artificial neural networks (ANNs) trained with the iris flower data as training set. As was shown on [16] and [21], the maximum fitness they can obtain is a success of 98.67%. With the system proposed here, we must be able to extract the knowledge contained in the ANNs not only in the training set, but also in new data points, because ANNs have the ability of generalization and once they are trained they accept as inputs values not present in the training set and are supposed to give good coherent outputs.

References

1. Koza J.: Genetic Programming. On the Programming of Computers by means of Natural Selection. The Mit Press, Cambridge Massachusetts (1992)
2. Montana, D.J.: Strongly Typed Genetic Programming. Evolutionary Computation. 3(2):199-200. The MIT Press, Cambridge Massachusetts (1995)
3. Darwin, C.: On the origin of species by means of natural selection or the preservation of favoured races in the struggle for life. Cambridge University Press. Cambridge, UK (1864).
4. Fuchs, M.: Crossover Versus Mutation: An Empirical and Theoretical Case Study. 3rd Annual Conference on Genetic Programming. Morgan-Kauffman (1998)
5. Luke, S., Spector, L.: A Revised Comparison of Crossover and Mutation in Genetic Programming. 3rd Annual Conference on Genetic Programming. Morgan-Kauffman (1998)
6. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press (1996)
7. Bonarini A.: Evolutionary Learning of Fuzzy Rules: Competition and Cooperation. Fuzzy Modelling: Paradigms and Practice. W. Pedrycz (Ed.), Kluwer Academic Press. Norwell, MA. (1996)
8. Koza J.: Genetic Programming II: Automatic Discovery of Reusable Programs. The Mit Press. Cambridge, Massachusetts (1994)
9. Wong, M.L., Leung, K.S.: Data Mining using Grammar Based Genetic Programming and Applications. Kluwer Academic Publishers (2000.)
10. Bot, M.: Application of Genetic Programming to Induction of Linear Classification Trees. Final Term Project Report, Vrije Universiteit, Amsterdam (1999)
11. Engelbrecht, A.P., Rouwhorst, S.E., Schoeman, L.: A Building Block Approach to Genetic Programming for Rule Discovery. Data Mining: A Heuristic Approach. Abbass, R. Sarkar, C. Newton editors, Idea Group Publishing (2001)
12. SAS Institute: SAS/STAT user's Guide, Release 6.03 Edition. SAS Institute Inc. Cary, North Carolina, U.S.A. (1988).
13. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Annals of Eugenic. 179-188. (1936)
14. Duch, W., Adamczak, R., Grabczewski, K.: A new methodology of extraction, optimisation and application of crisp and fuzzy logical rules. IEEE Transactions on Neural Networks, vol. 11, nº 2. (2000)
15. Kasabov, N.: Foundations of Neural Networks, Fuzzy Systems and Knowledge Engineering. MIT Press (1996)
16. Martínez, A., Goddard, J.: Definición de una red neuronal para clasificación por medio de un programa evolutivo. Mexican Journal of Biomedical Engineering. Vol. 22, pp. 4-11. (2001)
17. Browne, C., Düntsch, I., Gediga, G.: IRIS revisited: A comparison of discriminant and enhanced rough set data analysis. Polkowski L. and Skowron A. editors. Rough sets in Knowledge Discovery, vol. 2, Physica Verlag, Heidelberg, pp. 345-368. (1998)
18. Jagielska, I., Matthews, C., Whitfort, T.: The application of neural networks, fuzzy logic, genetic algorithms and rough sets to automated knowledge acquisition. 4th Int. Conf. On Soft Computing, IIZUKA’96. Japón, vol. 2, pp. 565-569 (1996)
19. Nauck, D., Nauck, U., Kruse, R.: Generating Classification Rules with the Neuro-Fuzzy System NEFCLASS. Proc. of Biennal Conf. of the North American Fuzzy Information Processing Society (NAFIPS’96). Berkeley. (1996)
20. Halgamuge, S.K., Glesner, M.: Neural Networks in designing fuzzy systems for real world applications. Fuzzy Sets and Systems. vol. 65, pp. 1-12 (1994)
21. Rabuñal Dopico, J.R.: Entrenamiento de Redes de Neuronas Artificiales mediante Algoritmos Genéticos. Graduate Thesis, Facultad de Informática, Universidade da Coruña (1999)