Application of a Genetic Programming Based Rule Discovery System to Recurring Miscarriage Data

Christian Setzkorn[1], Ray C. Paton[1], Leanne Bricker[2], and Roy G. Farquharson[2]

[1] Department of Computer Science, University of Liverpool, Chadwick Building, Peach Street, Liverpool L69 7ZF, United Kingdom
Email: chris@csc.liv.ac.uk (Setzkorn), rcp@csc.liv.ac.uk (Paton)
[2] Miscarriage Clinic, Liverpool Women's Hospital, Crown Street, Liverpool L8 7NJ, United Kingdom
Email: lbricker@dial.pipex.com(Bricker)

Abstract. This paper introduces a rule inference system based on the paradigm of genetic programming. Rules are deduced from a medical data set related to recurring miscarriage. A rule consists of an IF-part (antecedent) and a THEN-part (consequent). The system has to be supplied with the consequent and works out antecedents. An antecedent classifies the predictive class which is represented by the supplied consequent. The antecedents produced take the form of a tree, where Boolean operations such as AND, OR and NOT represent nodes, and Boolean expressions represent the leaves. Boolean expressions can be built from nominal and numeric attribute values, which makes the system very versatile.

1 Introduction

This paper introduces an approach developed during collaboration between the Computer Science Department of the University of Liverpool and the Recurrent Miscarriage Clinic (RMCL) of the Liverpool Women's Hospital. The approach uses the paradigm of Genetic Programming (GP) building upon the work of Freitas [3] [13]. The work can be described as an attempt to perform a generalised classification task for one prediction class which is represented by the consequent. The generalised classification task, also referred to as dependency modelling, has the goal of seeking to discover a few interesting rules called knowledge nuggets [13].

The objective of this collaboration was to produce computer-aided support tools that would help medical practitioners at the RMCL in their aim to gain new insights into the causes and unknown associations of an unfortunately common condition referred to as recurring miscarriage. Over the last ten years the staff at the RMCL have collected data from patients suffering from this

R.W. Brause and E. Hanisch (Eds.): ISMDA 2000, LNCS 1933, pp. 250–259, 2000.

condition and stored it as a spreadsheet; or rather a data matrix. In the data matrix a row contains data regarding a particular patient and the columns represent specific attributes. The attributes depict demographic details, information regarding previous pregnancies, other pertinent medical details, and observations made subsequent to referral to the RMCL.

The developed GP system has the capability of rule inference from data in the described data matrix form. The rules take the form of IF-THEN. The THEN-part (consequent) has to be supplied to the system and corresponds to a predictive class within the supplied data. The IF-part (antecedent) is evolved so that it classifies the predictive class as precisely as possible. An antecedent takes the form of a tree whose nodes may consist of Boolean operators: AND, OR and NOT. Boolean expressions represent the leaves of the tree. Examples of possible Boolean expressions are: $Age \geq 30$, *Blood Value X = abnormal* and *Occupation Level = High*.

This form of rule inference is a promising approach to evolve rules which have high prediction accuracy and represent new and interesting knowledge. In addition, evolved rules can be easily understood by users [5]. This, and the fact that the system can cope with nominal and numeric data, makes this approach especially attractive.

2 Genetic Programming - A Brief Introduction

John R. Koza introduced GP in 1987 with the objective of "enabling computers to solve problems without being explicitly programmed" [6] [7]. Numerous candidate solutions for a given problem are summarised and evolved within a population in a biologically inspired manner similar to genetic algorithm (GA) [8] [9] [10]. This allows a parallel search through the given problem domain in order to find one or more satisfactory solutions to a given problem.

Candidate solutions are also referred to as representation schemes or individuals, and are usually represented in the form of a tree with changeable size. A tree consists of so-called functions, which represent the nodes, and terminals representing leaves. Functions and terminals, available for building an individual, are respectively summarised within a function or a terminal set. The chosen sets must be suitable to the problem.

The more sophisticated tree representation scheme for GP reflects a crucial difference with GA's, where the representation scheme is normally unchangeable in size and form. This makes GP a more attractive and powerful candidate for the task of data mining due to the effectiveness of GP in searching very large spaces. On the other hand, the structure of a GP system itself is very similar to those of a GA. Figure 1 shows one such possible structure.

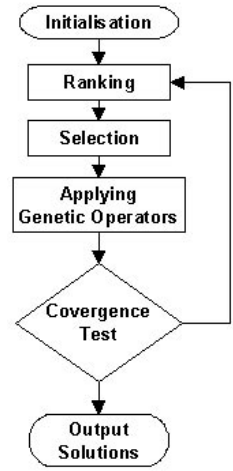

Fig. 1. Structure of a genetic algorithm.

During step 1 an initial population of candidate solutions is produced randomly. In step 2 each individual in this current population is evaluated with respect to its capability of representing a good solution for the given problem. An individual capability of solving the given problem is referred to as *fitness* and is calculated using an externally imposed fitness function, appropriate to the given problem. Step 3 consists of selecting individuals for the next generation according to their fitness. The selection process enables fitter individuals to be selected with a higher probability in comparison to individuals with lower fitness. There exist several selection processes such as roulette wheel selection, tournament selection, and ranking. [11] provides a summary and comparison of selection processes. Step 4 consists in employing the so-called genetic operators, such as crossover and mutation. The latter was not used in Koza's original form of GP. Figure 2 illustrates mutation in the case of GP.

Each individual, or rather its parts, has a specific probability of undergoing a mutation, determined by the parameter *mutation probability*. A random generator is used to decide whether or not a particular part of an individual undergoes mutation. If the value produced by the random generator is less than or equal to the *mutation probability* then the part of the tree is created anew using the supplied function and terminal set. Figure 3 illustrates crossover.

Two individuals are required in order to perform crossover in its simplest form. One possible implementation of crossover is to build a sub-population by randomly removing individuals from the current population. The choice of an individual for that sub-population works on the same principle as described for mutation, by using a random generator. Whether or not an individual is chosen is determined by the parameter *crossover probability*.

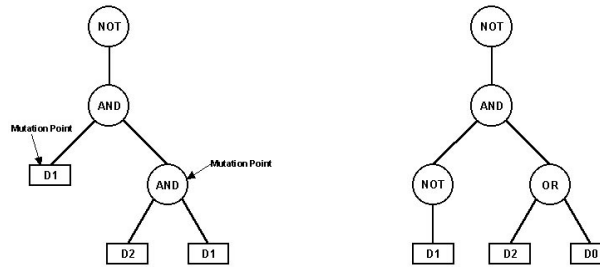

Fig. 2. Example of genetic operation mutation for genetic programming. On the left is an individual before mutation (parts to be mutated are marked), on the right of the figure is the result of the mutation.

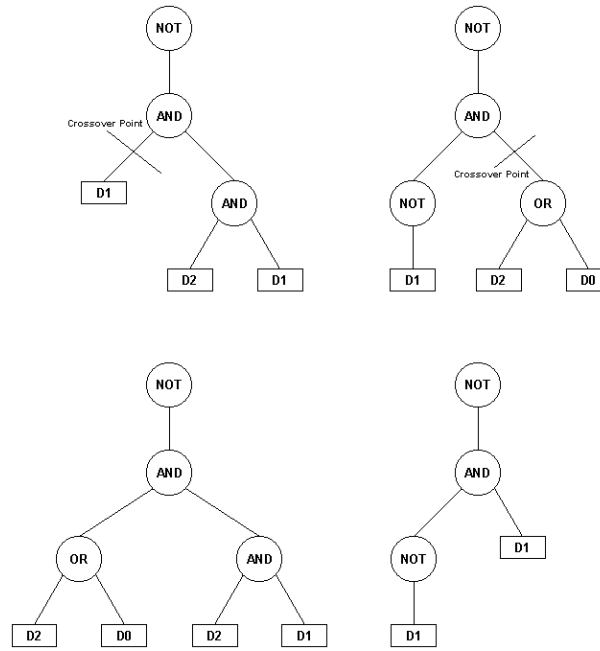

Fig. 3. Example of genetic operation crossover for genetic programming (D0 - D2 represent terminals).

Note, that it must be ensured that the sub-population has an even number of individuals. Consequently, individual pairs are taken from this sub-population, undergo crossover and are returned to the original population. For each individual a so-called crossover point is randomly determined, as shown at the top of figure 3. The parts below each crossover point are exchanged between the two individuals. The bottom of the same figure shows the result of crossover.

The next step is to perform the convergence test, which determines whether or not the system terminates and presents the individuals in the current population as solutions to the given problem, or evolves the population further by repeating steps 2 to 5 (see figure 1). A termination criterion may be a specific number of iterations or a specific fitness level to be achieved by the fittest individuals.

3 Details of the Implemented GP System

This section supplies some details of the implemented GP system. More information is available in [4].

Representation Scheme: An individual takes the form of a tree, where the function set may consist of three Boolean operations: *OR; AND; NOT*. The terminals take the form of Boolean expressions built depending on the supplied data set. A Boolean expression is composed of an attribute name, relational operator, and an attribute value. Three conditions are introduced which must be fulfilled during the creation of a tree: (1) Each attribute from the supplied data set can only be used once to build a Boolean expression. (2) The attribute(s) used to build the consequent must not be used to build a Boolean expression. (3) The depth of the tree is limited by the number of available attributes within the supplied data set minus the attribute(s) used for building the consequent. The user also can limit the depth of a tree.

These conditions and the nature of the supplied data set determine the composition of a Boolean expression. The composition is divided into three steps. The first step is determination of the attribute (name). The second step is random selection of one attribute value from all available values for the chosen attribute. The final step is the determination of the relational operator, which is constrained by the chosen attribute value. Obviously, if the available attribute values consist of nominal values, only the relational operator '=' would be appropriated. In a case where the attribute consists of numbers, other relational operators such as: $<; \leq; >; \geq; =$ are appropriated. However, there are two special cases. For numbers representing the smallest or greatest possible values of a particular attribute, the use of relational operators: $<; \leq$ and $>; \geq$ respectively would not be sensible.

Genetic Operators: Two genetic operators, crossover and mutation, were implemented in this study. These operators work on the same principle as described above. However, the maximum depth of a tree represents a restriction for crossover because insertion and/or exchange of sub-trees may cause an individual taking part in crossover to exceed its maximum depth. Also the inserted/exchanged sub-tree must not contain any attributes already contained within the recipient tree. Mutation can take place on functions and terminals. If a terminal mutates, it may create a new terminal (Boolean expression) or another function (Boolean operation) if the tree did not reach its maximum depth. Naturally, the building of a new Boolean expression (or several, if the terminal changes to a function) has the same restrictions as described above.

Fitness Measurement: The employed fitness function is based on the so-called *J-measure* [12]. The higher the *J-measure* for a particular individual, the higher its fitness. There are some weak points in the original *J-measure* as described in [13]. Due to this fact, this work used only the modified version of the *J-measure*. The modified version is presented as follows and referred to as *J1measure*.

$$\begin{cases} a = \dfrac{|P|}{N} \\ b = \dfrac{|C\&P|}{|C|} \\ J1measure = \dfrac{|C|}{N}\left(b \cdot log\left(\dfrac{b}{a}\right)\right) \end{cases} \tag{1}$$

Here C represents the number of cases in which the antecedent of a particular rule is fulfilled, whereas P describes the number of cases which fulfil the consequent. The expression $C\&P$ stands for the number of cases in which both the antecedent and the consequent are fulfilled at the same time. The resulting fitness function is:

$$fitness = \frac{w_1 \cdot (J1) + w_2 \cdot \left(\dfrac{n_{pu}}{n_T}\right)}{w_1 + w_2} \tag{2}$$

Where n_T is the number of attributes within the antecedent, n_{pu} is the number of potential useful attributes within the antecedent, and *J1* corresponds to the modified *J-measure*. An attribute is potentially useful, if the Boolean expression built from it, and the consequent, are fulfilled for at least one data entry at the same time. The values for w_1 and w_2 are user-defined weights and are assigned values of *0.6* for w_1 and *0.4* for w_2, as suggested in [13].

Selection: Three different selection mechanisms were implemented: roulette wheel selection, rank selection, and tournament selection. All three selection mechanisms produced similar results.

The Structure: The structure of the implemented GP system is the same as depicted in figure 1.

4 Experimental Investigations

The data set utilised during these experiments consisted of 353 complete cases (individual patients) and 13 attributes. These were obtained from the original data set consisting of many more cases (904) and attributes (50). Only specific attributes were selected. These were assumed to be valuable for the discovery of new knowledge regarding the causes and associations of recurring miscarriage. No feature selection has been employed so far. Since the original data set also contains incomplete cases, the number of actually available complete cases depends on the choice of attributes.

The data set was split into a training set and test set. The former contained two-thirds of the available cases and the latter the remaining third. It was ensured that each class was properly represented in both data sets. This is also referred to as stratification [16].

Figure 4 contains one example rule evolved for the consequent *Outcome Code* $= B$. It should be noted that the meaning of the presented rules and their Boolean expressions, cannot be explained in this paper. However, the data dictionary supplied in [4] can serve as a reference.

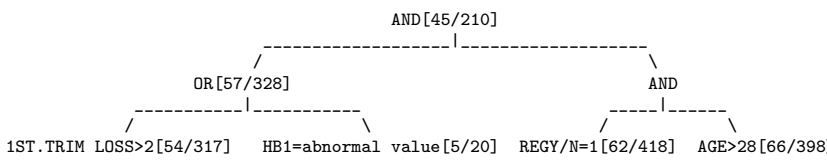

Fig. 4. Antecedent evolved for consequent **Outcome Code = B**.

In order to generate such a tree structure, a freely available program [15] was used since the rules produced by the implemented GP system were in form of s-expressions. S-expression are not easily comprehensible, but had to be used here due to lack of space.

Each node and leaf of the presented tree contains two numbers. The first number indicates how often the logical expression built by the particular part

of the tree and the logical expression built by the consequent are fulfilled in respect to the supplied data set at the same time. The second number indicates how often this is the case without considering the logical expression built by the consequent. These numbers are thought to be an additional comprehensible judgement of the quality of the particular tree and its parts.

Table 1 contains some of the best rules evolved by the GP system. A particular rule, in form of a s-expressions, was applied to the training and test data set and the value for *a* and *J1measure* (see formula 1 are supplied. In addition, the fitness achieved by a particular individual is also shown. The value of *a* is also referred to as *confidence factor* [13] and serves as a simple judgement of the performance of a particular rule applied to the unseen test data. The number *consequent hits* contains the number of cases in which the particular consequent is fulfilled in respect to the supplied data set.

Table 1. Certain results in the form of s-expressions.

Supplied Consequent	S-Expression	Fitness	J1measure	b	Consequnet Hits
APS = 0	(OR [139/200] (AND [48/64] (T4 1 = normal value [220/331]) (REG Y/N = 0 [53/71])) (AND [112/166] (OP1 = P [118/174]) (TSH 1 = normal value [219/329])))	0.4111	0.0186	0.695	234
	(OR [61/78] (AND [23/26] (T4 1 = normal value [100/144]) (REG Y/N = 0 [24/27])) (AND [49/66] (OP1 = P [51/71]) (TSH 1 = normal value [102/144])))	0.4241	0.0401	0.7821	112
APS = 1	(AND [94/259] (OR [110/322] (OP1 = S [63/179]) (2TL = 0 [96/284])) (REG Y/N = 1 [101/282]))	0.4118	0.0197	0.3629	119
	(AND [30/80] (OR [30/93] (OP1 = S [27/88]) (TSH 1 = abnormal value [5/15])) (REG Y/N = 1 [44/132]))	0.4269	0.0449	0.375	47
REG Y/N = 0	(AND [52/73] (OR [69/345] (TSH 1 = normal value [66/329]) (OP1 = S [34/179])) (DTNP1 = 0 [52/74]))	0.5118	0.1863	0.7123	71
	(AND [23/30] (OR [27/154] (TSH 1 = normal value [26/144]) (OP1 = S [13/88])) (DTNP1 = 0 [23/30]))	0.5308	0.2180	0.7666	27
REG Y/N = 1	(AND [255/272] (NOT [260/279] (DTNP1 = 0 [22/74])) (OR [276/345] (TSH 1 = normal value [263/329]) (OP1 = S [145/179])))	0.4694	0.1156	0.9375	282
	(AND [120/124] (NOT [125/129] (DTNP1 = 0 [7/30])) (OR [127/154] (TSH 1 = normal value [118/144]) (OP1 = S [75/88])))	0.4694	0.1157	0.9677	132

5 Conclusions and Further Work

The results demonstrated in this paper, yielded from the data set and the positive feedback obtained from the medical practitioners, led to the generation of conclusions regarding the system's practical potential. The presented results show that the rules perform well on unseen test data. Further research may reveal additional improvements in the quality of rules evolved by the system, and the system's performance.

Parallelisation of the GP system, as suggested in [13], is planned for future work. As well as enhancing the system's performance this may also lead to the discovery of qualitatively superior rules by using, for example, an *island model* as suggested in [2].

Furthermore, alternative fitness evaluation may prove more successful. In some cases the GP system is prone to developing rules of low interest value and thus truism knowledge. Ways to overcome this problem are suggested in [14].

Another possibility for improving the systems capability could be offered by *feature selection*. It results in an appropriate choice of features (attributes) for the current prediction class and thus a smaller search space for valuable rules.

More elaborated genetic operators might also prove to be more successful, as described in [4]. Other data sets have already been used to validate the developed system and proved its practical potential.

6 Acknowledgements

We would like to thank Dr. Alex A. Freitas for his support during the development of this approach.

References

1. Alex A. Freitas, Heitor S. Lopes, and Dieferson Luis Alves de Aranjo (1999). *A Parallel Genetic Algorithm for Rule Discovery in Large Databases*. Available at http://www.ppgia.pucpr.br/~alex/papers.html.
2. John R. Koza and David Andre (1995). *Parallel Genetic Programming on a Network of Transputers*. Available at ftp://elib.stanford.edu/pub/reports/cs/tr/95/1542/
3. Alex A. Freitas, Heitor S. Lopes, and Celia C. Bojarczuk (1999). *Discovering comprehensible classification rules using Genetic Programming: A case Study in a medical domain*. Available at http://www.ppgia.pucpr.br/~alex/papers.html.
4. Christian Setzkorn (2000). *Investigation into the Application of Artificial Intelligence Methods to the Analysis of Medical Data* . Available at http://www.csc.liv.ac.uk/~chris /Publications.html.

5. U. M. Fayyad, G. Piatetsky-Shapiro and P. Smyth (1996). From data mining to knowledge discovery: an overview. In: U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (eds.), Advanced in Knowledge Discovery & Data Mining, 1-34, AAAI/MIT
6. John R. Koza (1993). *Genetic Programming I.* MIT Press London
7. John R. Koza (1994). *Genetic Programming II.* MIT Press London
8. Lawrence Davis (1991). *Handbook of Genetic Algorithm.* Van Nostrand Reinhold; ISBN: 0442001738
9. Thomas Baeck (1999). *Evolutionary Algorithm in Theory and Practice.* Inst of Physics Pub; ISBN: 0750306653
10. K.F. Man, K.S. Tang, and S. Kwong (1999). *Genetic Algorithm.* Springer-Verlag Berlin
11. Tobias Blickle and Lothar Thiele (1995). *A Comparison of Selection Schemes used in Genetic Algorithm.* Available at http://www.handshake.de/user/blickle/publications/index.html
12. Smyth P., Goodman R. M. (1991). *Rule induction using information theory.* In Piatetsky-Shapiro G. and Frawley J. *Knowledge Discovery in Databases* MIT Press
13. Alex A. Freitas, Heitor S. Lopes, and Dieferson Luis Alves de Aranjo (1999). *A Parallel Genetic Algorithm for Rule Discovery in Large Databases.* Available at http://www.ppgia.pucpr.br/˜alex/papers.html.
14. Edgar Noda, Alex A. Freitas, and Heitor S. Lopes (1999). *Discovering Interesting Predictive Rules with a Genetic Algorithm.* Available at http://www.ppgia.pucpr.br//˜alex/papers.html.
15. Available at: http://www.dai.ed.ac.uk/daidb/students/chrisg/
16. Ian H. Witten and Eibe Frank (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers; ISBN: 1558605525