# Representing Classification Problems in Genetic Programming

**Thomas Loveard**
Department of Computer Science
Royal Melbourne Institute of Technology
GPO Box 2476V, Melbourne Victoria 3001, Australia
toml@cs.rmit.edu.au

**Victor Ciesielski**
Department of Computer Science
Royal Melbourne Institute of Technology
GPO Box 2476V, Melbourne Victoria 3001, Australia
vc@cs.rmit.edu.au

**Abstract- In this paper five alternative methods are proposed to perform multi-class classification tasks using genetic programming. These methods are: Binary decomposition, in which the problem is decomposed into a set of binary problems and standard genetic programming methods are applied; Static range selection, where the set of real values returned by a genetic program is divided into class boundaries using arbitrarily chosen division points; Dynamic range selection in which a subset of training examples are used to determine where, over the set of reals, class boundaries lie; Class enumeration which constructs programs similar in syntactic structure to a decision tree; and evidence accumulation which allows separate branches of the program to add to the certainty of any given class.**

**Results showed that the dynamic range selection method was well suited to the task of multi-class classification and was capable of producing classifiers more accurate than the other methods tried when comparable training times were allowed. Accuracy of the generated classifiers was comparable to alternative approaches over several datasets.**

## 1 Introduction

Genetic programming (GP) is a way of automatically constructing computer programs using a process analogous to biological evolution [6]. As a relatively new approach to problem solving, GP has been applied to a range of tasks including that of classification [3, 4, 5].

Due to the extremely flexible nature of computer programs to represent solutions to problems, GP methods of problem solving have a great advantage in their power to represent solutions to complex problems, and this advantage remains true in the domain of classification. This flexibility of representation gives genetic programs the capacity to represent classification problems with means unavailable to other commonly used classification techniques such as decision trees, statistical classifiers and neural networks. Because of this it is possible that GP methods could be more suited to some classification problems than other classification methods.

One advantage of using the GP methodology for classification tasks is that the more training time that is allowed the more accurate (to the point where over-fitting begins to occur) the classifier can be trained. If large amounts of time are available to train classifiers then this time can be utilised to better train genetic classifiers. This is an advantage over some other methods for performing classification (eg. C4.5), which will always produce the same classifier regardless of the amount of time available for training. Another advantage for GP is that each run is probabilistic and different runs attempting to solve the same problem will virtually never produce the same result. Such variability of the final solution lends GP classifiers well to a voting strategy, which is often able to produce more accurate classification results.

Of great importance to any programming language's capacity to represent a solution to a problem are the syntactic rules of the language and the set of primitive functions, types and operators in the language. This remains true for genetic programs, and when formulating such rules and primitives for a GP system for classification, the choice as to which rules and primitives to employ is one of uncertainty.

The aim of the research is to explore methods by which multi-class classification can be implemented within the GP paradigm and to identify which of these methods are most suited to the task of classification in terms of accuracy of classification and the amount of time needed to arrive at accurate solutions. Five alternative program representation methods, comprised of different syntactic rules, language primitives and decision strategies are developed and tested against a set of classification tasks.

### 1.1 Genetic Programming and Classification Tasks

Generally GP trees can perform classification by returning numeric (real) values and then translating these values into class labels [12, 13]. For binary classification problems the division between negative and non-negative numbers acts as a natural boundary for a division between two classes. This means that genetic programs can easily represent binary class problems. Unfortunately, when more than two classes are involved, finding meaningful division points over the set of reals the genetic programs return is more difficult. If boundary regions are chosen at arbitrary points over the set of reals then genetic programs face the problem of not only containing the necessary elements to distinguish between classes, but also must perform a translation task to provide output in the necessary range pre-specified for a given class.

Several investigations have previously employed a GP search strategy to generate a decision tree classifier [2, 9]. These investigations have shown the method to be capable

0-7803-6657-3/01/$10.00 ©2001 IEEE

of producing accurate classifiers. Such classifiers do however limit the produced programs to decision tree structures, which are more constrained (and thus less able to express certain problems) than standard genetic programs returning real values.

Previous investigations into classification using GP have produced accurate classifiers for binary class problems [3, 4]. GP employed for classification tasks do however have a requirement for long training times when compared to many other classification methods. It is also often quite difficult to extract a meaningful reason as to why a given class was chosen. Because of these factors the GP method is seen to be applicable to tasks where accuracy is the most important factor in classification, and training times and understandability are seen as relatively unimportant.

## 2 The Datasets

A set of six datasets were chosen from the UCI Machine Learning repository [1]. These datasets were chosen because they show variety in their domain, size and in the difficulty of classification. They also vary in the number of target classes for classification. All datasets were comprised of numeric or binary attributes. These datasets were also used in [7] and therefore allow direct comparison of results of GP classifiers to results of well known classification methods. The data sets are as follows:

1. Wisconsin Breast Cancer [8] (W.B.C): Consists of 2 classes and 10 numerical attributes with 699 instances. Sixteen instances containing missing values were removed for the purposes of classification in this investigation. Error rates were estimated using ten fold cross validation.

2. BUPA Liver Disorders (BUPA): Consists of 2 classes and 6 numerical attributes with 345 instances. Error rates were estimated using ten fold cross validation.

3. Pima Indians Diabetes (Pima): Consists of 2 classes and 8 numerical attributes with 768 instances. Error rates were estimated using ten fold cross validation.

4. Satlog Pixel (Pixel): Consists of 6 classes and 36 numerical attributes with 4435 instances. Error rates were estimated using an independent test set of 2000 instances.

5. Thyroid disease (Thyroid): Consists of 3 classes with 15 binary attributes (considered to be numerical for this investigation) and 6 numerical attributes. There were 3772 instances in the training set. Error rates were estimated using an independent test set of 3428 instances.

6. Vehicle silhouette (Vehicle): Consists of 4 classes and 18 numerical attributes with 846 instances. Error rates were estimated using ten fold cross validation.

## 3 Classifier Representation

Five alternative representation methods were implemented to produce solutions for the classification problems listed above. In each case strongly typed genetic programming [10] was used to construct genetic programs. For each method the terminal and function sets were kept as similar as possible so that the different GP methodologies had the same elements to construct solutions for the classification tasks. It was necessary to make some changes to the function and terminal set for the implementation of each representation method.

The five representation methods used were:

### 3.1 Binary Decomposition of Classification Problem

Standard methods of genetic program representation are able to perform binary classification tasks with a high degree of accuracy [4, 12]. Unfortunately, when more than two classes are involved, finding meaningful division points over the set of reals the genetic programs return, which then allow the output of a program to be interpreted as more than two classes, is more difficult. The problem of choosing arbitrary values can be avoided however by decomposing a multi-class problem into a set of binary classification problems [3, 4]. Given a problem $P$ with a set of $n$ classes $P = \{c1, c2, ...cn\}$, the problem can be decomposed into $n - 1$ binary classification problems. The first binary problem will be to distinguish between classes $\{c1, x\}$ where $x$ is an amalgamation of all other classes in the problem set. Ie: $P - \{c1\}$. The second problem will be to classify $\{c2, y\}$ where $y$ is all remaining classes $P - \{c1, c2\}$. This decomposition will continue until problem $n - 1$, which will consist of only $\{cn - 1, cn\}$, itself a binary classification problem.

Once solutions to all binary classification problems can be found, the amalgamation of these solutions forms a solution to the problem as a whole. The error rate of such a classifier would normally be found by applying the amalgamation of solutions to a test set. Due to the difficulty of attaining such an amalgamation initially in the GP methodology, in this implementation the error rate of the solution is calculated by adding the error rate of each component. Each binary solution was therefore weighted to account for only the section of the problem that it attempted to solve. The calculation of the error rate for the overall problem was:

$$E = \sum_{i=1}^{n-1} E(i) * \frac{|i|}{|P|}$$

where $i$ is a binary decomposition, $P$ is the overall problem set, $E(i)$ is the error rate of the solution for decomposition $i$, and $n$ is the number of distinct classes in the problem. This means that for the first binary decomposition the error rate is simply added to the overall error rate as |i| will be equal to |P|. The second decomposition will add only a proportion of its error rate to the overall error rate as all ele-

| Name | Return Type | Arguments | Argument Types | Description |
|---|---|---|---|---|
| Plus | Double | 2 | Double, Double | Arithmetic addition |
| Minus | Double | 2 | Double, Double | Arithmetic subtraction |
| Mult | Double | 2 | Double, Double | Arithmetic multiplication |
| Div | Double | 2 | Double, Double | Protected arithmetic division (divide by zero returns the value zero) |
| IF | Double | 3 | Boolean, Double, Double | Conditional. If arg1 is true, then return arg2, otherwise return arg3 |
| <= | Boolean | 2 | Double, Double | True if arg1 is less than or equal to arg2 |
| >= | Boolean | 2 | Double, Double | True if arg1 is greater than or equal to arg2 |
| = | Boolean | 2 | Double, Double | True if arg1 is equal to arg2 |
| Between | Boolean | 3 | Double, Double, Double | True if the value of arg1 is between the values of arg2 and arg3 |

Table 1: Standard Function Set

| Name | Return Type | Description |
|---|---|---|
| Random(-1, 1) | Double | Randomly assigned constant with value between -1 and 1 |
| Attribute[x] | Double | Value of attribute x |

Table 2: Standard Terminal Set

ments of the first class have already been classified (error is accounted for). This will continue until the error rate of each decomposition has been added to the overall error rate.

This method of calculation of the error rate will lead to a slight over-estimation of error in multi-class problems. For each decomposition errors will fall into two categories. These are either classifying an item of the target class as a class belonging to the remaining set of classes, or classifying an item from the remaining set of classes as an item of the target class. Over-estimation of the error occurs because of the second type of error. In a real classification system built using this method, an item of data that was misclassified at a higher level of decomposition but which actually belonged to a lower level of decomposition would not then be passed on to a lower level of decomposition for classification. In this investigation any item that is misclassified at a higher level of decomposition can still be misclassified at lower levels of decomposition. In this scenario however the true rate of error cannot be any higher than the recorded error rate and the amount of over-estimation is predicted to be minor. This is because, for most problems, error rates are small, meaning few items are misclassified, and thus the number of additional items that would be considered by classifiers at lower levels of decomposition is seen to be relatively small.

This approach uses the standard function and terminal sets shown in Table 1 and 2 respectively.

An example program built using this representation method can be seen in Figure 1. In this example the output of the tree is a real number. Due to the conditional node "IF", the output of the tree will either be the value of attribute 5 multiplied by: the constant -0.5; or attribute 2 added to attribute 7. If the constant value 0.23 is less than or equal to the value of attribute 2 then it will be the former case, otherwise the latter.

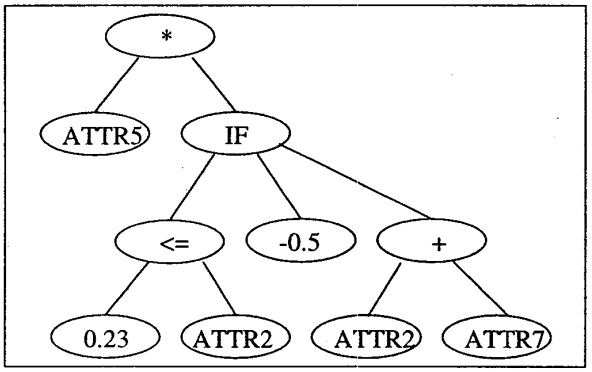


Figure 1: An example program tree for Binary Decomposition, Static Range Selection or Dynamic Range Selection

### 3.2 Static Range Selection

Genetic programs performing classification tasks generally return real values and, as stated above, it is difficult to find meaningful points over the set of real values to choose as division points for class boundaries, particularly for multi-class classification problems. In this approach we attempt to choose boundary regions based on intuitive guesses about the problem and likely points over the set of reals that class boundaries might occur.

For binary problems (W.B.C, BUPA and Pima) the choice for division is identical to that of the binary decomposition method, as the split between negative and non-negative numbers is used to distinguish between class boundaries.

The Pixel dataset (6 classes) used the following ranges: Class1 = [-inf, -5], Class2 = [-5, -1], Class3 = [-1, 0], Class4 = [0, 1], Class5 = [1, 5], Class6 = [5, inf].

For the Thyroid dataset (3 classes) the ranges used were:

Class1 = [-inf, -1], Class2 = [-1, 1], Class3 = [1, inf].

The Vehicle dataset (4 classes) used the following ranges: Class1 = [-inf, -1], Class2 = [-1, 0], Class3 = [0, 1], Class4 = [1, inf].

It is highly possible that the ranges chosen are not optimal for each problem, but it is intuitive that programs will be more readily able to produce results in these ranges because all randomly assigned constants are within the range [-1, 1] and all attributes are scaled between [0, 1]. Because it is not possible to know the optimal ranges however these selected values are as likely to perform well as any other values arbitrarily chosen.

This approach uses the standard function and terminal sets shown in Table 1 and 2 respectively.

### 3.3 Dynamic Range Selection

An alternative approach to static range selection, where ranges are arbitrarily chosen to correspond to class boundaries that all programs for the run must adhere to, is to allow each program to use a separate set of ranges for class boundaries that are dynamically determined for each individual program.

Given a classification problem with many training examples and an individual from a GP population it is possible to use a subset of the training examples and record the values that are returned when attributes for specific classes are used as inputs. Based upon these outputs the effectively infinite range of the reals can then be segmented into regions corresponding to class boundaries based upon areas the program has returned values for each class in the subset of training examples.

There would be many possible methods of segmenting the range of reals based on program outputs from a subset of training examples. For the purpose of practicality this investigation limited the range for segmentation to that of [-250,250]. This range is seen to be sufficiently large to allow programs to distinguish between all possible classes in the problem sets used in this investigation. The algorithm used in this investigation for dynamically determining the ranges for each program in the population is given in Figure 2. Note that the output value for each training example is rounded to the nearest integer. This rounding is performed to again limit the effectively infinite space of the real numbers, but is seen to still allow programs a necessary level of granularity over this range to distinguish between class boundaries.

Once segmentation of the output ranges has been performed the remainder of the training examples can then be used to determine the fitness of the given individual.

This approach uses the standard function and terminal sets shown in Table 1 and 2 respectively.

### 3.4 Class Enumeration

In this method a new data type is introduced called ClassType, which is an enumerated type. The set of values that this type

```
FOR each example X within the
    subset chosen for range selection {
  OUTPUT = execute program with X as input
  Round OUTPUT to nearest integer value
  IF OUTPUT < -250 THEN OUTPUT = -250
  IF OUTPUT > 250 THEN OUTPUT = 250
  Increment GP_OUTPUTS[OUTPUT][Class of X]
}

FOR COUNT = -250 to 250 {
  FOR all classes CL {
    IF all values for CL in the vector
       GP_OUTPUTS[COUNT][CL] are zero {
      RANGE[COUNT] = "?"
    } ELSE {
      Find CL for which
        GP_OUTPUTS[COUNT][CL] is greatest
      RANGE[COUNT] = CL
} } }

FOR COUNT = -250 to 250 {
  IF RANGE[COUNT] = "?" {
    RANGE[COUNT] = closest value to COUNT in
                   the RANGE vector whose
                   value is not "?"
} }
```

Figure 2: Pseudocode for Dynamic Range Selection for a Genetic Program

can store is limited to the number of different class types for the given problem. A new terminal, ClassNum is also introduced which returns a value of type ClassType, corresponding to one of the possible classes in the problem. Properties of this terminal can be seen in Table 4.

The program trees generated using this method of construction will always return a value of type ClassType. To facilitate the introduction of this new type, the "IF" function is modified (as per Table 3) to return a ClassType value, and the second and third argument are changed to accept only this type. This means that child sub-trees stemming from the second two arguments of an "IF" function can only hold another "IF" function, or a ClassNum terminal.

An example program can be seen in Figure 3 (where C$x$ refers to a ClassType terminal representing class $x$). Analysis of the structure of programs resulting from this method of construction shows a degree of similarity to the structure of decision tree classifiers such as C4.5 [11]. Execution of the program (in GP) or evaluation of the tree (in decision tree algorithms) will follow from the root node, through a series of conditional branches until a single leaf node, containing a class label, is reached. The major difference in the structure of the program trees produced by this GP approach to that of decision trees produced by C4.5 is that at each conditional branch of the tree the GP approach can consider almost any arithmetic expression that is possible given the function and terminal set. In contrast, conditional nodes of a C4.5 deci-

| Name | Return Type | Arguments | Argument Types | Description |
|---|---|---|---|---|
| IF | ClassType | 3 | Boolean, ClassType, ClassType | Conditional. If arg1 is true, then return arg2, otherwise return arg3. |

Table 3: Class Enumeration: Alteration to the Standard Function Set

| Name | Return Type | Description |
|---|---|---|
| ClassNum | Classtype | One, out of the possible set of classes for the given problem |

Table 4: Class Enumeration: Addition to the Standard Terminal Set

sion tree are only able to consider a single attribute value in comparison to a fixed constant value. This gives the GP program trees more freedom in their expressive capabilities, but also increases the search space to be considered by the genetic search.

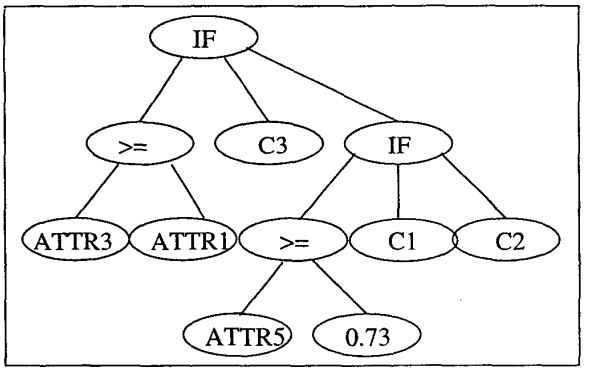


Figure 3: An example program tree for Class Enumeration

### 3.5 Evidence Accumulation

This method allows many different branches of the program trees to contribute to the decision to choose a certain class. Aside from the program tree each genetic program also contains a vector data storage area, termed the certainty vector. This vector contains one element for each possible class within the problem. Before program execution all elements of the certainty vector are initialised to zero. As the program executes values are added (or subtracted) from certain elements of the vector through the operation of a new terminal "AddToClass[x](-1, 1)" shown in Table 6. This terminal will add a value in the range -1 through 1, to one of the elements of the certainty vector.

When program execution halts, the certainty vector is examined and the element with the highest value in the vector is declared to be the most certain outcome for classification. In the event that two or more elements share the highest value, the results are inconclusive, and the outcome of classification is considered to be an error.

Because the outcome of the classification lies within the certainty vector it is not necessary for the program itself to return a value. This change in requirements for the program output means that an alteration to the conditional "IF" function must be made and an addition to the function set is also needed as shown in Table 5. The "BLOCK" function will accept from two to four arguments (this value is randomly chosen when programs are first generated) and simply evaluates each argument in a sequential order. This function is required so that it is possible for multiple additions (or subtractions) to the certainty vector are possible (with only conditional functions only one addition per program would be possible).

An example program can be seen if Figure 4 (terminals of the form $An(x)$ indicates an AddToClass function which will add to class $n$ the certainty value of $x$). In this example the program will automatically add the value 0.3 to the class 1 element of the certainty vector. Then, depending on whether attribute 3 is greater than or equal, or less than the value 0.6, it will either subtract 0.3, or add 0.7 to the certainty vector of class 3. The final classification will be the class with the greatest certainty value after execution.

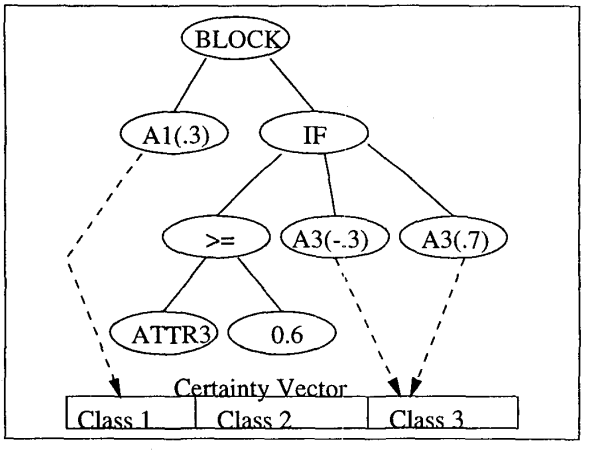


Figure 4: An example program tree for Evidence Accumulation

### 3.6 System Parameters

Due to the probabilistic nature of GPs, it is not certain that the outcome of one run will be the same as the output of any other run over the same training data. To help ensure valid results, each method was run ten times over each data set and the recorded error rates were then averaged over the ten runs.

| Name | Return Type | Arguments | Argument Types | Description |
|---|---|---|---|---|
| IF | NoValue | 3 | Boolean, NoValue, NoValue | Conditional. If arg1 is true then evaluate arg2, otherwise evaluate arg3. |
| BLOCK(2, 4) | NoValue | 2-4 | All NoValue | Evaluate all arguments sequentially. |

Table 5: Evidence Accumulation: Alteration and Addition to the Standard Function Set

| Name | Return Type | Description |
|---|---|---|
| AddToClass[x](-1, 1) | NoValue | Add a value between -1 and 1 to the certainty value of class x |

Table 6: Evidence Accumulation: Addition to the Standard Terminal Set

All numeric attributes accross the datasets were scaled to a range of 0 through 1. Each run consisted of a population size of 500 individuals, except for those against datasets with independent test sets for error evaluation. Population size for these two datasets (Pixel and Thyroid) was 1000 individuals. The termination criteria for each run was either perfect classification on the training set, or after 50 generations had been processed. When moving to a new generation elitist reproduction was used (10%), whilst the remaining 90% of individuals were selected for crossover using roulette wheel selection. No individuals were selected for mutation. Programs were generated with an initial maximum depth of 6, with overall program depth limited to 17.

Runs were conducted on a 4 processor ULTRA-SPARC4 and training times are given as a combination of user and system CPU time for the runs.

## 4 Summary and Analysis of Results

Tables 7 and 8 show the results obtained from the GP runs for test error rates and training times respectively. Classification error rates and times are given as an average of ten runs.

For the first three datasets (all binary classification problems: W.B.C., BUPA and Pima) there appeared to be little difference between the four methods of program representation. Dynamic range selection performed slightly better than any other method in terms of accuracy of classification, while class evaluation took slightly less time. The consistency of each method (shown in the standard deviation of error) varied only slightly over each dataset. Over all three datasets the evidence accumulation method required a great deal more run time. This is understandable, due to the "BLOCK" function leading to broader (and thus larger) programs and more evaluations of tree branches.

When the three multi-class problems were considered (Pixel, Thyroid and Vehicle) the variation in training time, accuracy, and consistency became much more distinct. The binary decomposition and dynamic range selection classifiers were far more accurate than the other three methods of classification, while the dynamic range selection method was more consistent (lower standard deviation of error). The binary decomposition method used a much longer period of time to produce the classifiers in these problems when compared to other methods, particularly in the case of the Vehicle and Pixel datasets (the Thyroid dataset can possibly be seen as a rare case amongst multi-class classification problems as 91% of instances belong to a singe class). The large amount of time required for training the binary decomposition classifiers for these multi-class problems is understandable, as several GP runs must occur to produce a single classifier, where methods like class evaluation and dynamic range selection can produce a classifier for the problem in a single run.

Results for the Pixel and Vehicle datasets indicated that dynamic range selection and class enumeration methods resulted in classifiers of relatively high accuracy in multi-class problems, but took less time to train than binary decomposition methods. Because of this it was of interest to know whether, given a greater amount of training time, such methods could exceed the classification accuracy of binary decomposition. Figure 5 shows the progression of test error rate as generations increased for the Pixel dataset for both the dynamic range selection and class enumeration methods. It can be seen that while dynamic range selection is able to immediately produce more accurate classifiers in the initial generation, it is slow to improve upon this fitness as generations progress. The class enumeration method begins with a less accurate starting point, but is able to utilise the genetic search process to improve fitness over time. However it can be seen that in the last ten generations the rate of improvement for the class enumeration and the dynamic range selection is very small, suggesting class enumeration would be unable to improve over the dynamic range selection method were more time allowed for training this method. To determine whether this was the case, and to determine whether, given comparable run time to the binary decomposition methods, these methods could outperform binary decomposition, the dynamic range selection and class enumeration methods were trialed again on the pixel and vehicle datasets. For the pixel dataset population sizes of 2000 were used, whilst for the vehicle dataset population sizes of 1000 were used. Runs were processed for 80 generations. The results of these runs showed that, when able to utilise the additional available run time, the dynamic range selection method was able to equal or surpass the binary decomposition method in terms of accuracy of classification on both datasets (error rate of 0.161 for the pixel dataset in 8:21:15, and error rate of 0.371 for the vehicle dataset in 6:18:53). The class enumeration method was however unable to surpass either binary decomposition

| | W.B.C. | | BUPA | | PIMA | | Pixel | | Thyroid | | Vehicle | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Mean | S.D. | Mean | S.D. | Mean | S.D. | Mean | S.D. | Mean | S.D. | Mean | S.D. |
| Binary Decomposition | 0.036 | 0.002 | 0.316 | 0.014 | 0.258 | 0.008 | 0.161 | 0.017 | 0.018 | 0.006 | 0.376 | 0.029 |
| Static Range Selection | 0.036 | 0.002 | 0.316 | 0.014 | 0.258 | 0.008 | 0.285 | 0.019 | 0.029 | 0.011 | 0.462 | 0.015 |
| Dynamic Range selection | 0.032 | 0.003 | 0.308 | 0.016 | 0.242 | 0.008 | 0.193 | 0.013 | 0.024 | 0.002 | 0.379 | 0.011 |
| Class Enumeration | 0.035 | 0.004 | 0.319 | 0.025 | 0.255 | 0.009 | 0.266 | 0.024 | 0.041 | 0.015 | 0.443 | 0.014 |
| Evidence Accumulation | 0.036 | 0.003 | 0.343 | 0.026 | 0.258 | 0.008 | 0.302 | 0.042 | 0.051 | 0.014 | 0.466 | 0.020 |

Table 7: Error Rate Mean and Standard Deviation for 10 Runs

| Method | W.B.C. | BUPA | PIMA | Pixel | Thyroid | Vehicle |
|---|---|---|---|---|---|---|
| Binary Decomposition | 2:27:59 | 1:16:29 | 1:56:55 | 8:50:12 | 1:56:01 | 6:22:56 |
| Static Range Selection | 2:27:59 | 1:16:29 | 1:56:55 | 1:45:18 | 2:00:43 | 2:00:06 |
| Dynamic Range selection | 2:16:37 | 1:13:07 | 1:35:56 | 2:30:28 | 1:44:24 | 2:00:14 |
| Class Enumeration | 1:50:31 | 1:04:11 | 1:27:51 | 1:37:30 | 0:49:28 | 1:47:54 |
| Evidence Accumulation | 5:07:52 | 3:17:03 | 4:09:54 | 3:47:49 | 2:19:50 | 6:07:08 |

Table 8: Mean Run Times (H:MM:SS) for 10 Runs

or dynamic range selection methods (error rate of 0.236 for the pixel dataset in 7:13:31, and error rate of 0.382 for the vehicle dataset in 6:56:53). These results indicate that the dynamic range selection method was more readily able to perform both multi-class and binary classification than any other method tried.

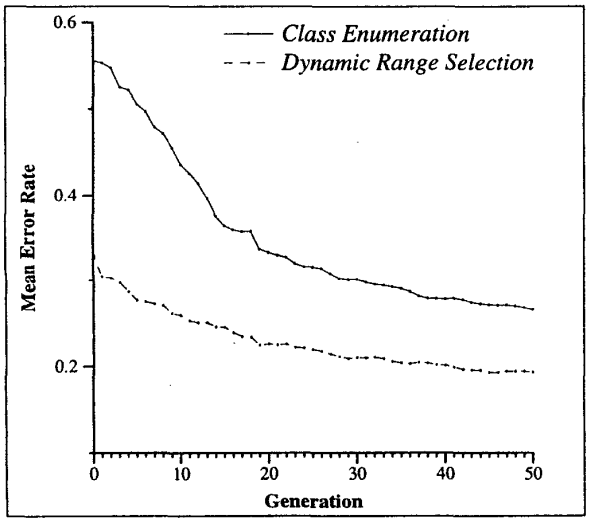


Figure 5: Error Rate for Best of Generation Individuals for the Pixel Dataset: Mean values over 10 Runs

The rank values given in Table 9 relate the results attained by the GP classifiers to thirty-three alternative methods of classification trained against these datasets in [7]. The rank values given are that of the classifier that would be displaced in rank (out of a possible of 33) by the genetic classifier. From these rank values it can be seen that for the W.B.C and BUPA datasets GP methods (and in particular dynamic range selection) are quite comparable to other contemporary methods of classification used in [7]. For other datasets, particularly the multi-class vehicle and pixel sets, it can be seen that the GP approaches tried here rank poorly in the set of thirty-three classifiers. However, in no case was a genetic classifier the worst performer out of the thirty-three (this would have resulted in a ranking of 34).

## 5 Conclusion and Future Work

The aim of this investigation was to explore alternative representations and implementations of multi-class classification problems within the GP paradigm, and determine which, out of five representation methods developed and tested, was better able to perform classification tasks. Each of the five methods resulted in classifiers that were competitive with the thirty three classifiers examined in [7]. Of the five methods developed, results indicate that dynamic range selection is more appropriate for both binary and multi-class problems as it is capable of producing classifiers of a higher degree of accuracy when comparable training times are allowed.

Results indicate that for some datasets GP classifiers are capable of producing competitive results, and in other datasets these classifiers perform relatively poorly when compared to other classification methodologies.

Future work, involving the use of subset selection techniques used in [4], dynamic selection of subsets used for the evaluation of ranges for the dynamic range selection method, larger populations and longer run times and the inclusion of automatically defined functions to GP classifiers, could substantially improve the performance of such GP classifiers when compared with other classification techniques. It is also possible to use the dynamic range selection method on mult-class datasets after these datasets have been decomposed into a set of binary datasets, as was performed in the binary decomposition method used in this paper. This could further improve the applicability of the dynamic range selection method to mulit-class problems.

Another future adaptation to the GP approach to classifi-

| Method | W.B.C. | BUPA | PIMA | Pixel | Thyroid | Vehicle |
|---|---|---|---|---|---|---|
| Binary Decomposition | 8 | 14 | 31 | 27 | 18 | 32 |
| Static Range Selection | 8 | 14 | 31 | 33 | 23 | 33 |
| Dynamic Range selection | 4 | 8 | 21 | 30 | 21 | 32 |
| Class Enumeration | 7 | 16 | 31 | 33 | 27 | 33 |
| Evidence Accumulation | 7 | 25 | 31 | 33 | 27 | 33 |

Table 9: Associated Ranks For Mean Error Rate Over 10 Runs

cation will be the inclusion of symbolic attribute information into GP classifiers.

## Acknowledgments

Thanks to the RMIT AI research group, in particular Ken Gardiner, Dylan Mawhinney, Paul Boxer, Andy Song and James Brusey for discussion and input into the ideas covered here. Thanks also to Peter Wilson whose genetic programming package was used for all GP runs.

## Bibliography

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] Martijn C. J. Bot and William B. Langdon. Application of genetic programming to induction of linear classification trees. In Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin and Terence C. Fogarty (editors), *Genetic Programming, Proceedings of EuroGP'2000*, Volume 1802 of *LNCS*, pages 247–258, Edinburgh, 15-16 April 2000. Springer-Verlag.

[3] J. Eggermont, A.E. Eiben and J.I. van Hemert. A comparison of genetic programming variants for data classification. In *Proceedings on the Third Symposium on Intelligent Data Analysis (IDA-99) LNCS 1642*, 1999.

[4] Chris Gathercole and Peter Ross. Dynamic training subset selection for supervised learning in genetic programming. In Yuval Davidor, Hans-Paul Schwefel and Reinhard Männer (editors), *Parallel Problem Solving from Nature III*, pages 312–321, Jerusalem, 9-14 October 1994. Springer-Verlag.

[5] Helen Gray. Genetic programming for classification of medical data. In John R. Koza (editor), *Late Breaking Papers at the 1997 Genetic Programming Conference*, page 291, Stanford University, CA, USA, 13–16 July 1997. Stanford Bookstore.

[6] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[7] T.-S. Lim, W.-Y. Loh and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning Journal*, Volume 40, pages 203–228, 2000. http://www.stat.wisc.edu/p/stat/ftp/pub/loh/treeprogs/quest1.7/.

[8] Olvi L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. Technical Report CS-TR-1990-958, University of Wisconsin, Madison, August 1990.

[9] Robert E. Marmelstein and Gary B. Lamont. Pattern classification using a hybrid genetic program decision tree approach. In John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba and Rick Riolo (editors), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 223–231, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[10] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, Volume 3, Number 2, pages 199–230, 1995.

[11] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992.

[12] Walter Alden Tackett. Genetic programming for feature discovery and image discrimination. In Stephanie Forrest (editor), *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 303–309, University of Illinois at Urbana-Champaign, 17-21 July 1993. Morgan Kaufmann.

[13] Mengjie Zhang and Victor Ciesielski. Genetic programming for multiple class object detection. In Norman Foo (editor), *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, Volume 1747, Lecture Notes in Artificial Intelligence, pages 180–191. Springer, Heidelberg, Dec 1999.