A Comparison of Genetic Programming Representations for Binary Data Classification

Emmanuel Dufourq
School of Mathematics, Statistics & Computer Science
University of KwaZulu-Natal
South Africa
edufourq@gmail.com

Nelishia Pillay
School of Mathematics, Statistics & Computer Science
University of KwaZulu-Natal
South Africa
pillayn32@ukzn.ac.za

***Abstract*—The choice of which representation to use when applying genetic programming (GP) to a problem is vital. Certain representations perform better than others and thus they should be selected wisely. This paper compares the three most commonly used GP representations for binary data classification problems, namely arithmetic trees, logical trees, and decision trees. Several different function sets were tested to determine which functions are more useful. The different representations were tested on eight data sets with different characteristics and the findings show that all three representations perform similarly in terms of classification accuracy. Decision trees obtained the highest training accuracy and logical trees obtained the highest test accuracy. In the context of GP and binary data classification the findings of this study show that any of the three representations can be used and a similar performance will be achieved. For certain data sets the arithmetic trees performed the best whereas the logical trees did not, and for the remaining data sets the logical tree performed best whereas the arithmetic tree did not.**

Keywords-data classficaition, genetic programming, data mining, optimization

I. INTRODUCTION

When implementing a genetic programming (GP) [15] algorithm for data classification the first thing a researcher has to consider is the representation. There are a vast number of existing representations which have been studied in the past however there exists three major ones, arithmetic trees, logical trees, and decision trees. GP is inspired by Darwin's theory of evolution. Each program is represented as an individual in a population. The individuals are encoded by a certain representation. GP in the domain of data classification has been successful. Researchers have been able to create models which obtain high classification accuracies [6, 18]. In the review of [18] several research works were surveyed for each of the three representations. There were a total of 24 studies for GP and binary classification using arithmetic trees, and a total of 11 studies which made use of logical trees. There were also a vast number of studies which used decision trees. Similarly in the review of [6] there were a vast number of papers surveyed for each of the three representations. In the previous studies researchers provided no justification for their choice of representation. This leads to an immediate question, why was that representation selected over another? Does a particular representation result in models which obtain higher accuracies? These questions are the rationale behind this study.

This study compares the three major representations in the context of GP and binary data classification. Different function sets were used to determine which combination of functions would yield better results. The representations were tested on eight publicly available data sets.

The rest of the paper is organized as follows. Section II presents the rationale for this research. This is followed by section III which presents the relevant literature and a description on how each representation is investigated in this study. Section IV describes the experimental setup. The results are presented and discussed in section V, and finally section VI concludes this paper.

II. RATIONALE

Researchers have successfully used GP to solve data classification problems using either arithmetic trees, decision trees, or logical trees. There is no doubt as to the capabilities of GP in the domain of data classification to create models which can obtain high accuracies. However there is a research gap when it comes to a comparison of GP representations for binary data classification and is the rationale driving this study. A new researcher entering the field of GP and binary data classification will have to decide which representation to use. Such a decision can be difficult to make without a comparison of the available representations. In previous studies researchers do not state a reason for their choice of representations. In certain cases authors will state that a representation was used because of its apparent capabilities of handling the data. Such a reason can be seen in [12] whereby they argue that since most data sets consist of numerical data it is appropriate to make use of arithmetic functions rather than Boolean functions. The objectives of this study are listed as follows:

- Implement the arithmetic tree, decision tree, and the logical tree representations for GP and binary data classification.
- Compare their performance in terms of classification accuracy and the size of models.

978-1-4799-3230-6/13/$31.00 ©2013 IEEE

- To determine if a particular representation results in a higher classification accuracy when compared to the others.

This will enable new researchers to have a greater understanding of which representation is more suitable for binary data classification.

III. GP REPRESENTATIONS FOR DATA CLASSIFICATION

A. Arithmetic trees

Arithmetic trees represent mathematic expressions which can discriminate between classes. In the case of binary classification one tree can discriminate between two classes using a threshold value. The function set consists of mathematic operators such as +, -, *, /, log, tan, exp. Leaf nodes represent attributes whereas the non-left nodes are the mathematical operators. A tree represents a single mathematical expression which outputs a single real valued number. The output is then compared to a threshold value. Assume a classification problem has classes "*a"* and *"b"*. If the output is less than the threshold value, then the class value "*a"* is the resulting classification value for that particular tree. If the output is greater than or equal to the threshold value, then the class value "*b"* is the classification value. Thus in this representation the tree does not directly encode the class value. Figure 1 illustrates an example of a GP arithmetic tree for data classification. In the example "x20" represents attribute 20. There are two mathematical functions in this example, the addition and multiplication operators. The tree represents the following expression (x20 * x4) + x15.

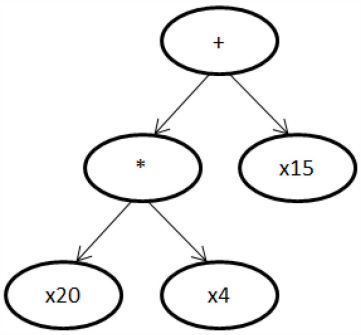

Figure 1. Arithmetic tree representation for GP.

Etemadi et al. [4] apply a different approach to a binary classification problem. A 0/1 rounding threshold with a rounding threshold value of 0.5 was applied. When a tree outputted a value greater of equal to the rounding threshold the instance of data was classified as bankrupt firm, otherwise the instance was classified as non-bankrupt firm. The function set consisted of {+, -, *, ^, NOT, LT}. The NOT operator has an arity of one and returns the result obtained by subtracting the argument from 1, and the LT operator has an arity of 2 and returns a value of 1 if the first argument is smaller than the second variable, otherwise the operator returns 0. The researchers do not mention how the rounding threshold value of 0.5 was obtained.

Gray et al. [5] make use of 20 *Varimax* scores as terminals to evolve trees to classify brain tumors. The class values +1 and -1 used to represent non-meningioma and meningioma instances. The function set consisted of {+, -, *, /, myAND, myOR, myNOT, tan} where the logical operators returns either 0 or 1 based on their logical evaluation. The researchers point out the tan function was not present in the best individual and that only three functions were present. A threshold value of 0 was used to determine the class of an individual. A positive output corresponded to +1, and a negative output corresponded to -1.

A simple threshold value of 0 is used in [20] to distinguish between the two classes for binary classification problems. In addition a small function set consisting of {+, -, *, /, if} was used. Hennessy et al. [9] investigate the use of GP for the binary classification task of determining if a solvent is present or absent in a mixture of solvents in a Raman spectra. The data set used consisted of 1024 attributes and only 24 instances. Each GP individual was represented by an arithmetic expression consisting of only {+, -} operators. A threshold value of 0 was used to map the output of an individual to either presence, or absence of a solvent.

In our study a threshold value of zero is used similarly to other methods found in previous studies. An *if* function was added into the function set to allow for conditional checks. The function takes three parameters. The first is a Boolean check and this check is achieved using either the less than equality or the greater than equality operator. The GP algorithm is free to create any sub-tree for the first parameter provided the top most node is an inequality node. If the first parameter results in a value of "*true*", then the second branch is traversed, and if the first parameter results in a value of "*false*" the third branch is traversed. The *if* function has been included into the function set in previous studies [17, 20, 21]. However no comparisons have been made in order to determine the usefulness of the function. Certain studies [22, 23] do not include the *if* function.

B. Decision Trees

Decision trees do not represent an expression like arithmetic trees or logical trees do. Decision trees represent a path from the root node to one leaf node. Each node within the tree represents one attribute and the leaf nodes represent one class value. When a decision tree is traversed a choice as to which branch will be visited next is determined by the attribute. Figure 2 illustrates a simple decision tree. The root node is the attribute *temperature*. Thus for any instance in a data set if the temperature value is *hot*, the left node is visited next, if the temperature value is *cold*, then the right node is visited next. The class values are directly encoded into the tree. In figure 2, when the left branch is visited a

leaf node is reached and the classification output value is "*class 1*".

Tur and Guvenir [3] used genetic programming to evolve decision trees. The fitness function used considers both the size in terms of the number of nodes and the accuracy of the classifier, furthermore the function makes use of weights to select which of the size or accuracy has a greater impact on the fitness of a tree. The method was applied to a single binary classification problem. Koza [14] presents how GP was used to create decision trees in solving the Saturday Morning problem presented by Quinlan.

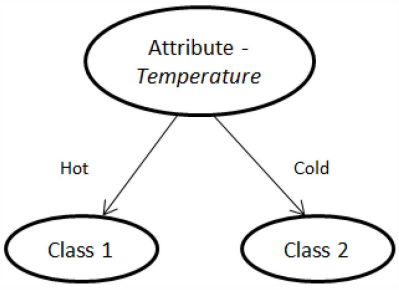

Figure 2. Decision tree representation.

The attribute in figure 2 is categorical and in such a case a branch is created for each possible value for the attributes. However when dealing with continuous data such an approach is infeasible as this will result in trees have a large number of branches. This will create a large program space and will hinder the performance of the GP algorithm. Discretization is used to overcome this problem as is defined as the process of transforming a continuous attribute into a discrete one with a finite number of intervals [7, 19]. In this study an adaptive discretization technique is used to create decision trees which can handle continuous data. Adaptive discretization was introduced by Bacardit and Garrell [24]. Their proposed method was tested on eight data sets. The adaptive approach obtained higher classification accuracies on six data sets when compared to a simple uniform width interval approach. Bacardit and Garell further improved the research in [25]. When a node is created the GP algorithm randomly creates two intervals. Figure 3 presents an example of two randomly created intervals for some attribute, in this case the attribute data ranged from 0.0 to 6.0. The first interval ranges from [0.0 to 3.25) and the second interval from [3.25 to 6.0]. The GP algorithm is free to create any two random intervals.

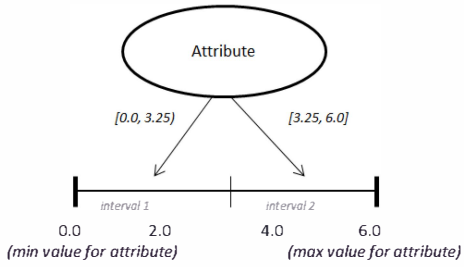

Figure 3. Illustrating two randomly created intervals

A genetic operator was implemented in order to evolve the intervals dynamically within the GP evolution. At each generation when the tournament selection is applied to obtain a parent, a random node is then selected from that tree. The two intervals for that particular node are randomly created ten times. If the classification accuracy of the tree is improved during any of those ten attempts then this process is stopped and the genetic operator inserts the modified tree into the population.

C. *Logical trees*

Logical trees represent a logical expression which is constructed from several logical operators. The function set consists of the logical operators and is typically made up of the AND, OR, and the NOT operators. Inequality operators are included into the function to create a comparison between values and return a Boolean value. Similarly to arithmetic trees this representation does not directly encode the class values into the tree.

Kuo et al. [1] evolve tree structures using GP. The fitness function makes use of the accuracy and the complexity the tree. The complexity of a tree is defined as the ratio of the nodes in the tree compared to those in the initial trees. The function set consisted of {AND, OR, NOT, >, ⩾ , <, ≤ , If-Then, If-Then-Else}. The terminals consisted of the values that each attribute could take as well the class values. Two operators were developed to resolve the issues of redundancy and subsumption. The eliminator operator dealt with identifying sub-trees within a single tree which encode the same rule, if two sub-trees were found then the one deepest in the tree was removed. The merge operator attempted to remove rules that subsumes another by examining if two rules had the same classes and if the features of one was a subset of the other. The proposed method was applied to a binary classification problem and was compared against the performance of C5.0 and a standard GP algorithm. The performance of the proposed method outperformed the other two.

De Falco et al. [10] investigate the use of classification rules by making use of trees. The GP algorithm creates an initial population and evolves the population over several generations. This is repeated until a classifier is found for each class. The function set consisted of {AND, OR, NOT, <, ⩽, >, ⩾, IN, OUT}. The IN and OUT functions take 3 arguments and are used to represent internal and external intervals. The terminal set consisted of the attributes. The authors developed a distance measure to resolve instances which resulted in a clash when an instance of data was allocated to more than one class or in the case whereby an instance was not allocated to any class.

Logical trees can be created for each class and a tree should output a true value for instances belonging to that class and a false values for those which do not belong to that class. Thus if there are two classes, the GP algorithm will result in two trees, one for each class. The shortcomings of this approach occurs when two trees output a true value for

a particular instance which means that the instance belongs to both classes. Similarly if two trees output a value of false for an instance then the instance belongs to neither of the two classes. When using such an approach of creating a tree for each class, a mechanism for dealing with clashes has to be incorporated.

In this study a similar approach to the arithmetic representations was used. Assume a classification problem has classes "*a"* and *"b"*. If a tree outputs a true value, then the classification output is class "*a"*, and if the tree outputs a false value then the output is class "*b"*. This allows a single tree to discriminate between two classes in a similar manner to arithmetic trees. This approach also avoids the extra complexity of dealing with clashes. This proposed approach has not been investigated in previous studies.

The terminal set consists of the attribute values. In this study random constant values were added to the terminal set. These random values were generated between the minimum and maximum value which exists in the training data which the GP algorithm takes in as input

Figure 4 illustrates an example of a logical tree. In this example the tree first checks if x20 is less than x4. Then the tree checks if x8 is greater than x12. The OR operator is applied to the result of the two comparisons.

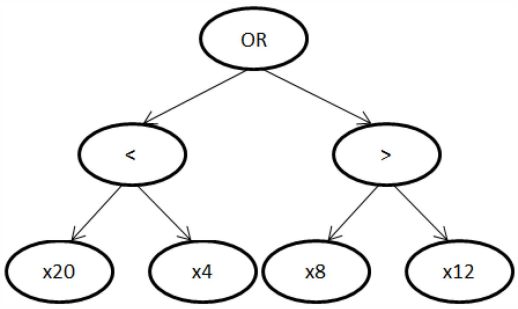

Figure 4. Logical tree representation for GP.

Typically inequality functions are used with this representation. A *between* function was proposed in order to allow an attribute to be compared to two values simultaneously. Figure 5 illustrates an example of the between node.

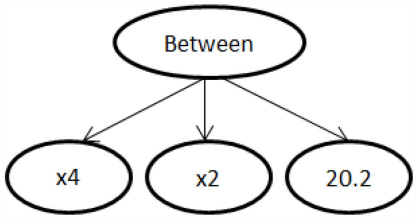

Figure 5. The between GP operator for logical tree representations.

The *between* function takes three parameters of which the first parameter is always an attribute and the other two can be attributes or constants. The function is defined as follows:

$$\text{Between}(x1, y1, y2) = \begin{cases} \text{true} & \text{if } y1 \leqslant x1 \leqslant y2, \\ \text{false} & \text{otherwise.} \end{cases}$$

A similar function to the *between* function was used by De Falco *et al.* [10]. They created two functions, IN and OUT which compare an attribute value with two range values. These functions also take three parameters. The OUT function checks if an attribute value is outside of the range of the two parameters. In this study the *OUT* function can be achieved when a *NOT* function precedes the *between* function. However in the study of De Falco *et al.* they do not determine whether or not these additional functions are useful in evolving classifiers.

D. Additional representations

There exist several other representations which are found in the literature. Jabeen and Baig [26] present a novel approach for dealing with mixed attributes for binary classification. The proposed method consists of a two layer approach, an outer layer and an inner layer. The outer layer conists of functions {AND, OR, NOT} and their terminals correspond to the inner layer trees. The researchers allow for two kinds of inner layers, one being a logical inner layer and the other an arithmetic inner layer. The logical inner layer consists of functions {AND, OR, NOT} and the terminals consist of the categorical attributes attached with an "=" or "≠" sign, thus a terminal could be C1=' a'. The arithmetic inner layer makes use of {+, -, *, /} functions and the terminals correspond to the numerical attributes. The proposed method was tested on five datasets yielded better results than methods that use different approaches for handling mixed attributes.

In the work of [27], a two stage learning approach makes use of arithmetic expressions to create classifiers for classes within a dataset. A population of trees is created for each class. The first stage of their proposed method consists of creating the populations of trees for each class. In this stage the function set consisted of {+, -, /, *} and the terminal set consisted of the real attributes and an ephemeral constant. The second stage of the proposed method consists of creating a chromosome which corresponds to each classifier for all the classes. Thus if there are n classes, the chromosome has length n where each gene represents a classifier for each class. A new population of chromosomes is created in this stage by applying the tournament selection to the populations created in the first stage. The proposed method was tested on 5 datasets and achieved better results when compared to a binary decomposition method.

IV. EXPERIMENTAL SETUP

A. Experiments

Five experiments were carried out in order to compare the three different represents but also to compare how certain elements of the function set impact the performance of the GP algorithm. Table I lists the five different experiments along with their function set and terminal set. The GP

algorithm is run several times on all the data sets, this is further discussed in the following section. The experiment using the "*arithmetic tree with If*" representation was performed to determine the usefulness of the *if* statement. Similarly the experiment using the *"logical tree without Bet"* representation was performed to investigate effectiveness of the *between* function. The GP algorithm was programmed in Java 6. Each GP run was executed on the CHPC's cluster [2] and was limited to 4 CPU cores per run.

TABLE I. FIVE DIFFERENT EXPERIMENTS TESTED.

Representation	Function set	Terminal set
Arithmetic tree 1	+, -, *, /	Attributes
Arithmetic tree with If	+, -, *, /, if, <, >	
Decision tree	Attributes	Class values
Logical tree 1	AND, OR, NOT, BET, <, >	Attributes and random constants
Logical tree without Bet	AND, OR, NOT, <, >	

B. Data sets and model validation

Table II presents the eight data sets which were used in this study. These data sets were selected due to their popularity in binary data classification research and because of their differences in characteristics. The climate model simulation crashes (Climate) and fertility data sets were selected due to their recent inclusion to the repository. These data sets were obtained from the UCI Machine Learning Repository [8].

TABLE II. CHARACTERISTICS OF THE DATA SETS.

Data set	Number of instances	Number of attributes	Class balance
Climate	540	18	8.52 / 91.48
Fertility	100	9	88.00 / 12.00
Ionosphere	351	34	35.90 / 64.10
Parkinsons	195	22	75.38 / 24.62
Pima Indians	768	8	65.10 / 34.90
Sonar	208	60	53.37 / 46.63
Spectf	267	44	79.40 / 20.60
WDBC	569	32	62.74 / 37.26

In order to examine the success of the experiments and to remain consistent to methods found in the literature the 10-cross fold validation [11, 13, 16] method was used. This involves separating the data into ten random folds of equal size. When the GP algorithm is run the first fold is used for testing and the remaining nine folds are used for training. The GP algorithm is run again this time with the second fold as a test set and once again the remaining folds are used for training. This is repeated in such a way that each fold is used exactly once for testing. When the data is separated into ten folds it is possible that some of the folds contain data which can impact the learning phase. For this reason the entire process is repeated five times. At each repetition the data is separated into ten new random folds. For each iteration exactly one fold is used for testing as described and in turn each fold is used as a test set once. On each of the data sets the GP algorithm was run a total of fifty times and each execution of the GP algorithm was run on a unique random seed.

C. GP parameters

The GP parameters used through all the experiments are listed in table III. These parameters were determined empirically through trial runs. Parameter tuning was performed by investigating different values for the population size, tournament selection size, maximum offspring size, and the maximum number of generations. The crossover and mutation rates were selected to be 70% and 30% respectively as those values are widely used in literature. Allowing the GP algorithm to run beyond 200 generations did not necessarily improve the accuracy of the classifiers. Amongst the different initial population generation methods, namely the full method and the grow method, it has been observed that in the existing literature across numerous application domains that the ramped half and half method results in a diverse initial population which permits the GP algorithm to potentially search the correct program space.

TABLE III. GP PARAMETERS USED IN THIS STUDY

GP Parameter	Value
Population Size	700
Parent Selection Method	Tournament selection, size: 7
Maximum Initial Population Tree Size	7
Initial Population Generation Method	Ramped half and half
Maximum Offspring Size	7
Crossover	70
Mutation	30
Maximum Number of Generations	200
GP Control Model	Generational model

V. RESULTS AND DISCUSSION

The 10 cross-fold validation results for the experiments are presented in table IV. From the five representations tested the arithmetic trees and logical trees without the *between* function are the two methods which obtained the highest results over several data sets. The arithmetic tree ranked 1st in four data sets and the logic tree without the between function ranked 1st in three data sets. The arithmetic trees with the *if* function and the decision trees did not rank 1st in any data set. The logical tree representation ranked 1st only in the *Ionosphere* data set.

The results obtained by for the *Ionosphere*, *Sonar*, *Pima* and *Fertility* data sets were of interest. For those four data sets the arithmetic tree representation performed poorly in comparison to the performance of both the logical representations. The two logical representations obtained the highest accuracy for those data sets when compared to the other representations.

TABLE IV. CLASSIFICATION ACCURACY RESULTS (%) OF EXPERIMENTS FOR THE EIGHT DATA SETS. THE AVERAGE TEST RESULTS ARE PRESENTED. THE 10-CROSSFOLD VALIDATION WAS USED. VALUES IN BOLD REPRESENT THE BEST RESULT FOR A DATA SET.

Representations		Climate	Iono	Park	Pima	Fertility	Sonar	Spectf	WDBC
Arithmetic trees	Train	97.42	95.83	91.99	74.94	95.20	87.84	86.62	97.20
	Test	**94.33**	88.71	**86.70**	69.30	82.00	72.47	**77.77**	**95.15**
	Size	*26.88*	*35.56*	*29.12*	*41.08*	*26.64*	*30.28*	*33.16*	*25.72*
Arithmetic trees with if	Train	97.43	95.97	92.24	75.79	95.96	90.05	88.31	97.28
	Test	94.15	88.94	84.72	69.29	82.20	74.30	76.87	94.76
	Size	*28.42*	*37.10*	*33.12*	*49.20*	*28.34*	*38.48*	*43.46*	*32.70*
Logical trees	Train	96.64	98.06	89.85	77.94	94.09	94.04	91.96	95.32
	Test	92.15	**92.02**	84.83	73.67	84.20	75.16	75.97	92.76
	Size	*33.94*	*45.18*	*23.68*	31.90	28.36	*50.36*	*45.72*	*30.84*
Logical trees without between	Train	97.19	97.96	89.61	78.75	93.40	93.88	89.58	95.60
	Test	92.78	92.01	85.17	**73.96**	**84.80**	**75.28**	76.83	92.51
	Size	*37.72*	*41.90*	*21.04*	43.14	21.54	*53.30*	*36.68*	*30.06*
Decision trees	Train	96.86	96.42	95.31	81.92	95.87	94.70	91.15	97.52
	Test	90.78	90.93	86.69	73.62	80.20	73.00	76.58	93.99
	Size	*35.32*	*25.04*	*26.00*	51.24	19.76	*47.60*	*38.60*	*28.84*

However the complete opposite observation can be made for the remaining data sets. For the *Spectf*, *WDBC*, *Climate*, and *Parkinson's* data sets the arithmetic representation obtained the highest accuracy. This was not always the case for the arithmetic tree representation with the *if* function. For those four data sets the arithmetic representation obtained higher results than the arithmetic representation with the *if* function. Additionally when the *if* function was added the average tree size was always larger than without the *if* function. There was no data set whereby any of the arithmetic representations and logical representations obtained high results simultaneously. The results show that whenever one of the two arithmetic representations performed well, one of the logical representations would not perform as well.

TABLE V. TRAINING AND TEST AVERAGE RESULTS (%) ACROSS ALL THE DATA SETS.

	Train		Test	
Representations	Accuracy	Rank	Accuracy	Rank
Arithmetic trees	90.88 ± *1.30*	5	83.30 ± *6.47*	3
Arithmetic trees with if	91.63 ± *1.51*	4	83.15 ± *6.97*	5
Logical trees	92.24 ± *2.31*	2	83.84 ± *6.70*	2
Logical trees without between	92.00 ± *2.10*	3	**84.17** ± *6.72*	1
Decision trees	**93.72** ± *1.49*	1	83.22 ± *6.70*	4

There are no apparent characteristics within the data sets that would stand out as a possible reason for this difference in performance between the arithmetic and logical trees. Arithmetic trees perform well in the *WDBC*, *Spectf*, *Parkinson's* and the *Climate* data sets. These have attributes varying from 18 to 44 and the sizes of the data sets vary from 195 to 569. The minority class percentage varies from 8.52 to 37.26. The logical trees achieve good results in the *Sonar*, *Ionosphere*, *Fertility* and *Pima* data sets.

In those data sets the attribute values range from 8 to 60 and the size of the data sets vary from 100 to 768. The minority class value ranges from 12.00 to 46.63. The characteristics of the eight data sets are very similar in nature and cannot be the cause for such a difference in performance.

The decision tree representation performed well in certain data sets and poorly in others. There was no observable trend similar to that found with the arithmetic and logical representations.

Decision trees worked well in both the *Parkinson's* and *Pima* data sets. The decision trees obtained a slightly lower performance to the best result on the *Parkinson's* data set by 0.01%, and a lower performance to the best result for the *Pima* data set by 0.36%. The decision tree representation obtained the highest overall accuracy for the training data sets. Table V presents both the training and testing average accuracy across all the data sets. A ranking is also provided whereby a smaller number denotes a better performing method.

VI. CONCLUSION

This study compared the performance of three major representations for GP and binary data classification. Several variations of these representations were proposed in order to determine which function set would yield better results. The representations were tested on eight data sets.

Based on the findings from this study when GP is being used for binary data classification a researcher may pick any of the three representations and obtain good results. However due to the slightly weaker performance of decision trees this representation is not recommended. Additionally this representation requires discretization. Logical tree representations with the function set {AND, OR, NOT, <,

>} and arithmetic trees with the function set {+, -, *, /} generally perform well and thus these representations are recommended. Arithmetic trees with the *if* function does not improve the performance and thus is not necessary. Similarly for the logical trees, the *between* function does not improve the overall performance. The rationale behind this study was that researchers did not provide sufficient justification for their choice of representations. Furthermore a researcher would not want to select a representation that would perform poorly. This study empirically shows that researchers investigating the domain of GP and binary data classification can select an arithmetic or logical representation without being concerned about the performance of either of those two representations.

There was no consistency between the arithmetic and logical representations. When one performed well the other did not, and thus future research will aim at determining the cause for such an observation. Future research also includes extending this study to multiclass classification problems in order to determine which representation is the most suitable.

ACKNOWLEDGMENT

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF. The authors would like to thank the CHPC [2] for granting access to their resources.

REFERENCES

[1] C.-S. Kuo, T.-P. Hong, and C.-L. Chen, "Applying genetic programming technique in classification trees," *Soft Computing*, vol. 11, no. 12, pp. 1165–1172, 2007.

[2] Centre for high performance computing [Online]. Available: http://www.chpc.ac.za/

[3] G. Tur and H. A. Guvenir, "Decision tree induction using genetic programming," in *Proceedings of the Fifth Turkish Symposium on Artificial Intelligence and Neural Networks*.

[4] H. Etemadi, A. A. Anvary Rostamy, and H. F. Dehkordi, "A genetic programming model for bankruptcy prediction: Empirical evidence from iran," *Expert Systems with Applications*, vol. 36, no. 2, pp. 3199–3207, 2009.

[5] H. Gray, R. Maxwell, I. Martinez-Perez, C. Arus, and S. Cerdan, "Genetic programming for classification of brain tumours from nuclear magnetic resonance biopsy spectra," *Genetic Programming*, p. 424, 1996.

[6] H. Jabeen and A. R. Baig, "Review of classification using genetic programming," *International journal of engineering science and technology*, vol. 2, no. 2, pp. 94–103, 2010.

[7] H. Liu, F. Hussain, C. L. Tan, and M. Dash, "Discretization: An enabling technique," *Data mining and knowledge discovery*, vol. 6, no. 4, pp. 393–423, 2002.

[8] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[9] K. Hennessy, M. G. Madden, J. Conroy, and A. G. Ryder, "An improved genetic programming technique for the classification of raman spectra," *Know.-Based Syst.*, vol. 18, no. 4-5, pp. 217–224, Aug. 2005.

[10] I. De Falco, A. Della Cioppa, and E. Tarantino, "Discovering interesting classification rules with genetic programming," *Applied Soft Computing*, vol. 1, no. 4, pp. 257–269, 2002.

[11] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[12] J. Fitzgerald and C. Ryan, "Exploring boundaries: optimising individual class boundaries for binary classification problem," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, ser. GECCO '12*. New York, NY, USA: ACM, 2012, pp. 743–750.

[13] K. J. Cios, L. A. Kurgan, W. Pedrycz, and R. W. Swiniarski, *Data Mining: A Knowledge Discovery Approach*. Springer Science+ Business Media, LLC, 2007.

[14] J. R. Koza, "Concept formation and decision tree induction using the genetic programming paradigm," in *Parallel Problem Solving from Nature*. Springer, 1991, pp. 124–128.

[15] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press, 1992.

[16] M. Bramer, *Principles of data mining*. Springer, 2007.

[17] W. Smart and M. Zhang, "Classification strategies for image classification in genetic programming," in *Proceeding of image and vision computing conference*, pp. 402–407, Palmerston North, New Zealand, 2003.

[18] P. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *Systems, Man, and Cybernetics, Part C: Applications and Reviews*, IEEE Transactions on, vol. 40, no. 2, pp. 121–144, 2010.

[19] S. Garcia, J. Luengo, J. Saez, V. Lopez, and F. Herrera, "A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 25, no. 4, pp. 734–750, 2013.

[20] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," 2011.

[21] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 2, pp. 1070–1077, IEEE, 2001.

[22] B.-C. Chien, J.-y. Lin, and W.-P. Yang, "A classification tree based on discriminant functions," *Journal of information science and engineering*, vol. 22, no. 3, p. 573, 2006.

[23] H. Jabeen and A. R. Baig, "Two-stage learning for multi-class classification using genetic programming," *Neurocomputing*, 2013.

[24] J. Bacardit and J. M. Garrell, "Evolution of multi-adaptive discretization intervals for a rule-based genetic learning system," in *Advances in Artificial Intelligence–IBERAMIA 2002*, pp. 350–360, Springer, 2002.

[25] J. Bacardit and J. M. Garrell, "Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system," in *Proceedings of the 2003 international conference on Genetic and evolutionary computation: PartII, ser. GECCO'03*, (Berlin, Heidelberg), pp. 1818–1831, Springer-Verlag, 2003.

[26] H. Jabeen and A. R. Baig, "Two layered genetic programming for mixed attribute data classification," *Appl. Soft Computing.*, vol. 12, pp. 416–422, Jan. 2012.

[27] H. Jabeen and A. R. Baig, "Two-stage learning for multi-class classification using genetic programming," *Neurocomputing*, 2013.