Classification Strategies for Image Classification in Genetic Programming

Will R. Smart, Mengjie Zhang
School of Mathematical and Computing Sciences,
Victoria University of Wellington, New Zealand
{smartwill, mengjie}@mcs.vuw.ac.nz

Abstract

This paper describes an approach to the use of genetic programming for multi-class image recognition problems. In this approach, the terminal set is constructed with image pixel statistics, the function set consists of arithmetic and conditional operators, and the fitness function is based on classification accuracy in the training set. Rather than using fixed static thresholds as boundaries to distinguish between different classes, this approach introduces two dynamic methods of classification, namely centred dynamic range selection and slotted dynamic range selection, based on the returned value of an evolved genetic program where the boundaries between different classes can be dynamically determined during the evolutionary process. The two dynamic methods are applied to five image datasets of classification problems of increasing difficulty and are compared with the commonly used static range selection method. The results suggest that, while the static boundary selection method works well on relatively easy binary or tertiary image classification problems with class labels arranged in the natural order, the two dynamic range selection methods outperform the static method for more difficult, multiple class problems.

Keywords: genetic algorithms, object recognition, centred dynamic range selection, slotted dynamic range selection, genetic programs.

1 Introduction

Genetic programming (GP) is a relatively recent and fast developing approach to automatic programming [1, 2]. In genetic programming, solutions to a problem are represented as computer programs. Darwinian principles of natural selection and recombination are used to evolve a population of programs towards an effective solution to specific problems. The flexibility and expressiveness of computer program representation, combined with the powerful capabilities of evolutionary search, make GP an exciting new method to solve a great variety of problems.

Since the 1990s, GP has been applied to some real world classification problems including detecting and recognising particular classes of objects in images [3, 4, 5, 6, 7, 8]. In these systems, GP classifiers model a solution to a classification problem in the form of a mathematical expression, using a set of arithmetic and mathematical operators, possibly combined with conditional/logic operators such as the "if-then-else" structures commonly used in computer programs.

The output of a GP classifier is a numeric value that is typically translated into a class label. For the simple binary classification case, this translation can be based on the sign of the numeric value [4, 9, 10, 11]; for multi-class problems, finding the appropriate boundary values to separate the different classes is more difficult. The simplest approach (*static range selection*) – fixing the boundary values at manually chosen points – often results in unnecessarily complex programs and could lead to poor performance and very long training times [6, 8].

The goal of this paper is to develop new classification strategies in GP for multi-class image classification problems. The main focus is on the translation of the numeric output of a genetic program classifier into class labels. Rather than using manually pre-defined boundary values, we will consider methods which allow each genetic program to use a set of dynamically determined class boundaries. We will compare the two methods with the current static (manually defined) method on a number of image classification problems of increasing difficulty.

This paper is organised as follows. Section 2 describes the overall GP approach for object classification problems. Section 3 describes the two classification strategies. Section 4 presents the five image classification problems used in this approach. Section 5 presents the experimental results and section 6 gives the concluding remarks.

2 GP Applied to Object Classification

In this approach, we used the based tree-structure to represent genetic programs [12]. The ramped half-and-half method was used for generating the programs

in the initial population and for the mutation operator [1]. The proportional selection mechanism and the reproduction [8], crossover and mutation operators were used in the learning and evolutionary process [1].

In the remainder of this section, we address the other aspects of the GP learning/evolutionary system: (1) Determination of the terminal set; (2) Determination of the function set; (3) Construction of the fitness measure; and (4) Selection of the input parameters and determination of the termination strategy.

2.1 Terminals

For object classification problems, terminals generally correspond to image features. Some conventional approaches to image recognition usually use high level, domain specific features of images as inputs to a learning/classification system, which generally involves a time consuming feature selection and a hand-crafting of feature extraction programs. In this approach, we used pixel level, domain independent statistical features (referred to as *pixel statistics*) as terminals and we expect the GP evolutionary process can automatically select features that are relevant to a particular domain to construct good genetic programs.

Four pixel statistics are used in this approach: the average intensity of the whole object cutout image, the variance of intensity of the whole object cutout image, the average intensity of the central local region, and the variance of intensity of the central local region.

Since the range of these four features are quite different, we linearly normalised these feature values into the range [-1, 1] based on all object image examples to be classified.

In addition, we also used some constants as terminals. These constants are randomly generated using a uniform distribution. To be consistent with the feature terminals, we also set the range of the constants as [-1, 1]. Unlike the feature terminals where the same feature usually has different values for different object images, the constant terminals will remain unchanged for all object images through the whole evolutionary process.

2.2 Functions

In the function set, the four standard arithmetic and a conditional operation was used to form the function set:

$$FuncSet = \{+, -, *, /, if\} \quad (1)$$

The $+$, $-$, and $*$ operators have their usual meanings — addition, subtraction and multiplication, while / represents "protected" division which is the usual division operator except that a divide by zero gives a result of zero. Each of these functions takes two arguments. The *if* function takes three arguments. The first argument, which can be any expression, constitutes the condition. If the first argument is negative, the *if* function returns its second argument; otherwise, it returns its third argument. The *if* function allows a program to contain a different expression in different regions of the feature space, and allows discontinuous programs, rather than insisting on smooth functions.

2.3 Fitness Function

We used classification accuracy on the training set of object cutout images as the fitness function. The classification accuracy of a genetic program classifier refers to the number of object cutout images that are correctly classified by the genetic program classifier as a proportion of the total number of object images in the training set. According to this design, the best fitness is 100%, meaning that all object images have been correctly recognised.

To calculate the classification accuracy of a genetic program, one needs to determine how to translate the program output into a class label. This is described in section 3.

2.4 Parameters and Termination Criteria

The parameter values used in this approach are shown in table 1.

In this approach, the learning/evolutionary process is terminated when one of the following conditions is met:

- The classification problem has been solved on the training set, that is, all objects of interest in the training set have been correctly classified without any missing objects or false alarms for any class.
- The number of generations reaches the predefined number, *max-generations*.

3 Classification Strategies

As mentioned earlier, each evolved genetic program has a numeric output value, which needs to be translated into class labels. The methods which perform the translation are referred to as *classification strategies* in this paper.

This section briefly describes the static range selection (SRS) method for multi-class classification commonly used in many approaches, then details the two new classification strategies: centred dynamic range selection (CDRS) and slotted dynamic range selection (SDRS).

Table 1: Parameters used for GP training for the three databases.

Parameter Kinds	Parameter Names	Shape1	shape2	coin1	coin2	coin3
Search Parameters	population-size	300	300	300	500	500
	initial-max-depth	3	3	3	3	3
	max-depth	5	5	5	6	6
	max-generations	50	50	50	50	50
	object-size	16×16	16×16	70×70	70×70	70×70
Genetic Parameters	reproduction-rate	20%	20%	20%	20%	20%
	cross-rate	50%	50%	50%	50%	50%
	mutation-rate	30%	30%	30%	30%	30%
	cross-term	15%	15%	15%	15%	15%
	cross-func	85%	85%	85%	85%	85%

3.1 Static Range Selection

Introduced in [5, 6], the static range selection (SRS) method has been used in many approaches to classification problems with three or more classes. In this method, two or more pre-defined thresholds/boundaries are applied to the numeric output value of the genetic program and linearly translated the ranges/regions between these boundaries as different classes. This method is simple because these regions are set by the fixed boundaries at the beginning of evolution and remain constant during evolution.

If there are N classes in a classification task, these classes are sequentially assigned to N regions along the numeric output value space from some negative numbers to positive numbers by N−1 thresholds/boundaries. Class 1 is allocated to the region with all numbers less than the first boundary, class 2 is allocated to all numbers between the first and the second boundaries, and class N to the region with all numbers greater than the last boundary N−1.

3.2 Centred Dynamic Range Selection

The first method we developed is the Centred Dynamic Range Selection (CDRS), where the class boundaries are dynamically determined by calculating the centre of the program output values for each class. The algorithm is presented as follow.

Step 1 Initialise the class boundaries as some random values as in the SRS method.

Step 2 Evaluate each genetic program in the population to obtain the program output result value for each training example and the fitness value of the program based on the fitness function.

Step 3 For each class c, calculate the centre of the class according to equation 2:

$$\text{Center}_c = \frac{\sum_{p=1}^{M} \sum_{\mu_c=1}^{L} (W_p \times \text{Result}_{p\mu_c})}{\sum_{p=1}^{M} \sum_{\mu_c=1}^{L} W_p} \quad (2)$$

where M is the number of programs in the population and p is the index, L is the number of total number of training examples for class c and μ_c is the index, $Result_{p\mu_c}$ is the the output value of the pth program on training example μ_c for class c, and W_p is a weight factor which reflect the relative importance or contribution of the program p over all the programs in the population and is calculated by equation 3:

$$W_p = fitness_p + 50\% \quad (3)$$

where $fitness_p$ is the fitness (classification accuracy) of program p on all the examples in the training set.

Step 4 Calculate the boundary between every two classes by taking the middle point of the two adjacent class centres.

Step 5 Classify the training examples based on the class boundaries and calculate the new fitness (classification accuracy) of each genetic program.

3.3 Slotted Dynamic Range Selection

The second new classification strategy is Slotted Dynamic Range Selection (SDRS). In this method, the output value of a program is split into certain slots. In our experiment, we used 100 slots derived from the range of [-25, 25] with a step of 0.5. Since the input features (terminals) are scaled into [-1, 1], the range [-25, 25] is usually sufficient to represent the program output. Each slot will be assigned to a value for each class.

In the first step, this method evaluates each genetic program in the population to obtain the program output

value (`Progout`) for each training example and the fitness value of the program based on the fitness function and SRS method.

In the second step, the method calculates the slot values for each class (`Array[slot][class]`) based on the program output value and the fitness value. The algorithm for this step is as follows:

```
FOR each slot and each class
   Array[slot][class] = 0
FOR each training example X {
   FOR each program p {
      ProgOut = execute program p with X as input
      Round ProgOut to nearest slot
      IF Progout > 25 THEN Progout = 25
      IF Progout < -25 THEN Progout = -25
      Array[slot][class] += W_p
   }
}
```

Where W_p is calculated according to equation 3, reflecting the relative contribution of the genetic program over all the programs.

In the third step, this method dynamically determines to which class each slot belongs by simply taking the class with the largest value at the slot. However in case a slot does not hold any value, that is, no programs produce any outputs at that slot for any training examples, then this slot will be assigned to the class of the nearest neighbouring slot, as shown in the following algorithm:

```
FOR slot = 1 to 100 {
   FOR all class c {
      IF all values of class c in
            Array[slot][c] are zero {
         Range[slot] = ''?''
      }
      ELSE {
         Search c for which
            Array[slot][c] is largest
            Range[slot] = c
      }
   }
}

FOR slot = 1 to 100 {
   IF Range[slot] = ''?'' {
      Range[slot] = nearest value to slot
         in the Range vector whose value
         is not ''?''
   }
}
```

It is important to note that these methods are applied to the evolutionary process every five generations so that at other generations the programs will be only updated based on the evolutionary beam search.

4 Image Data Sets

We used four image databases in two groups in the experiments. Example images are shown in figure 1.

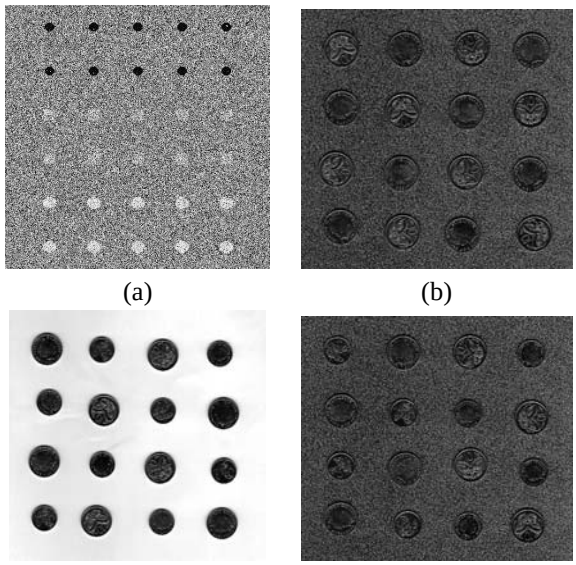

Figure 1: Example images from Shapes (a), Coin1 (b), Coin2 (c) and Coin3 (d).

4.1 Computer Generated Shape Datasets

The first group of images (figure 1 (a)) was generated to give well defined objects against a noisy background. The pixels of the objects were produced using a Gaussian generator with different means and variances for each class. Four classes of 713 small objects were cut out from these images to form the classification data. The four classes are: black circles, light grey squares, white circles, and grey noisy background.

Two different data sets, *shape1* and *shape2* were constructed from this group of images. While set shape1 arranges the four classes in a natural order based on the intensities, set shape2 out of this order.

4.2 NZ Coin Datasets

The second group of images has three NZ coin datasets. These datasets were intended to be harder than group 1 and consist of scanned 5 cent and/or 10 cent New Zealand coins. In this group, three data sets, *coin1*, *coin2* and *coin3*, were constructed to provide object classification problems of increasing difficulty. Example images for each of the three datasets are shown in figure 1 (b), (c), and (d), respectively. The first coin data set has 576 object cutout images of three classes: 10 cent heads, 10c tails, and a noisy background. The second coin data set consists of five classes of object cutouts: 5 cent heads, 5 cent tails, 10 cent heads and 10 cent tails, and a relatively uniform background. The third coin data set also consists of five classes of object cutouts, but the background is highly clustered, which makes the classification problems much harder.

The objects in each of the these data sets were equally split into three separate data sets: one third for the training set used directly for learning the genetic program

classifiers, one third for the validation set for controlling overfitting, and one third for the test set for measuring the performance of the learned program classifiers.

5 Results and Discussion

This section presents a series of results of the two dynamic classification strategies on the five data sets in the shape and coin image groups. These results are compared with those for the static classification method. For all experiments, we run 10 times and the average results were presented.

5.1 Shape Data Sets

Table 2 shows the results of the three methods on the two shape data sets. The first line shows that for the Shape1 data set with 4 classes, the SRS method achieved an average accuracy of 99.72% of 10 runs on the test set and the average number of generations of the 10 runs spent on the training process was 11.3.

Table 2: Results on the shape data sets.

Data set	Classes	Method	Gens	Accuracy
Shape1	4	SRS	11.3	99.72%
		CDRS	11.9	99.67%
		SDRS	20.5	98.06%
Shape2	4	SRS	47.0	95.52%
		CDRS	18.4	98.35%
		SDRS	29.8	99.53%

For the Shape1 data set, all the three classification methods obtained nearly ideal results, reflecting the fact that this classification problem is relatively easy. In particular, the SRS method achieved the best performance.

For the Shape2 data set, the two dynamic methods gave very good results. While the SDRS method resulted in better results than the CDRS method, it took a longer time to learn. In particular, the static SRS method produced a much worse performance in both classification accuracy and training time than the two dynamic methods because the classes in the data set were arranged arbitrary rather than in a natural order. This suggests that while the SRS method can perform well on relatively easy classification problems with the classes arranged in a natural order, this method is not appropriate for multi-class object classification problems with an randomly arranged order of classes. The two dynamic methods, however, can be applied in this case.

5.2 Coin Data Sets

Table 3 shows the results of the three methods on the three coin databases.

Table 3: Results on the coin data sets.

Data set	Classes	Method	Gens	Accuracy
Coin1	3	SRS	4.9	99.63%
		CDRS	7.4	99.89%
		SDRS	16.4	99.63%
Coin2	5	SRS	50	85.82%
		CDRS	50	88.10%
		SDRS	50	87.41%
Coin3	5	SRS	50	72.67%
		CDRS	50	78.17%
		SDRS	47.3	77.83%

For data set Coin1, while all the three methods achieved almost ideal performance, the CDRS gave the best classification accuracy and the SRS method used the shortest training time. For the second and the third data sets Coin2 and Coin3, the two dynamic methods clearly outperformed the static SRS method.

As expected, the classification performance on the three coin datasets deteriorated as the degree of difficulty of the object classification problems increased.

5.3 Summary and Discussion

In summary, the results suggest that the SRS method can perform well when there are a small number of object classes and the classes are arranged in their natural order (such as Shape1 and Coin1), but would perform badly when the classes are out of this natural order (as in Shape2) or when the classification problems become more difficult or the number of classes become more than four (such Coin2 and Coin3). The main reason is that a high degree of non-linearity is required to map the class regions on the program output to the object features if the classes are arranged in different orders. One would also expect SRS to perform well on two-class problems.

The performances of all the three methods on the Coin2 and Coin3 were worse than the Coin1 and the two shape classification problems. Because these problems were more difficult, more features should be selected, extracted and added to the terminal set. Also more powerful functions can also be applied in order to obtain good performance. However, the investigation of these developments is beyond the goal and the scope of this paper.

6 Conclusions

The goal of this paper was to investigate and explore dynamic classification methods in genetic programming for multi-class object classification problems, and to determine whether the dynamic methods could outperform the current static method for relatively difficult problems. Two dynamic

methods, CDRS and SDRS, were developed and implemented where the class boundaries were dynamically determined during the evolutionary process.

The results on the five object classification problems in two groups of images showed that the commonly used static method, SRS, performed very well when there were only a small number of object classes and the classes are arranged in their natural order, but performed badly when the classes were arranged in random order. The two dynamic methods, CDRS and SDRS, performed better than the static SRS method on most of the object classification tasks described here and this is particularly true when the problems were getting harder or the number of classes became larger. They generally took longer training times. However, if a much better performance can be obtained, this price is worth to pay.

These results suggest that, for binary or tertiary object classification problems with classes in natural order, both the static method SRS and the two dynamic methods, CDRS and SDRS, can be applied, and the static method SRS is recommended if training time is a critical factor. For other situations, particularly for relatively difficult problems or when there are a large number of object classes, the two dynamic methods are recommended.

For future work, we will investigate whether the performance on the relatively difficult coin data sets can be improved if more features are added to the terminal set. We will also investigate the power and reliability of the two dynamic methods on even more difficult, real-world image classification problems such as face recognition problems and satellite image detection problems, and compare the performance with other long-term established methods such as decision trees, neural networks, and support vector machines.

7 Acknowledgements

We would like to thank Dr Peter Andreae at VUW for a number of useful discussions.

References

[1] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of computer programs and its Applications*. San Francisco, Calif. : Morgan Kaufmann Publishers; Heidelburg : Dpunktverlag, 1998. Subject: Genetic programming (Computer science); ISBN: 1-55860-510-X.

[2] John R. Koza. *Genetic programming : on the programming of computers by means of natural selection*. Cambridge, Mass. : MIT Press, London, England, 1992.

[3] Thomas Loveard and Victor Ciesielski. Representing classifi cation problems in genetic programming. In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1070–1077, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 May 2001. IEEE Press.

[4] Andy Song, Vic Ciesielski, and Hugh Williams. Texture classifi ers generated by genetic programming. In David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 243–248. IEEE Press, 2002.

[5] 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.

[6] 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 (AI'99)*, pages 180–192, Sydney, Australia, December 1999. Springer-Verlag Berlin Heidelberg. Lecture Notes in Artifi cial Intelligence (LNAI Volume 1747).

[7] Mengjie Zhang, Peter Andreae, and Mark Pritchard. Pixel statistics and false alarm area in genetic programming for object detection. In Stefano Cagnoni, editor, *Applications of Evolutionary Computing, Lecture Notes in Computer Science, LNCS Vol. 2611*, pages 455–466. Springer-Verlag, 2003.

[8] Mengjie Zhang, Victor Ciesielski, and Peter Andreae. A domain independent window-approach to multiclass object detection using genetic programming. *EURIASP Journal on Signal Processing, Special Issue on Genetic and Evolutionary Computation for Signal Processing and Image Analysis*, 2003(8):841–859, 2003.

[9] Daniel Howard, Simon C. Roberts, and Richard Brankin. Target detection in SAR imagery by genetic programming. *Advances in Engineering Software*, 30:303–311, 1999.

[10] Jamie R. Sherrah, Robert E. Bogner, and Abdesselam Bouzerdoum. The evolutionary pre-processor: Automatic feature extraction for supervised classifi cation using genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 304–312, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[11] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, Faculty of the Graduate School, University of Southern California, Canoga Park, California, USA, April 1994.

[12] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge, Mass. : MIT Press, London, England, 1994.