Canonical linear GP for classification

csci6506: Sandbox 1

Due date: September 28 – submit on BrightSpace

Task definition

- Your goal is to develop an implementation for canonical GP for multi-class classification using a *linear GP* representation.
- It is recommended that you first read the paper by [1],[1] however, as a baseline implementation I would not expect you to incorporate all the properties described in this paper. Hence, although 'Removing introns at runtime' is definitely useful, support for this in your code is not necessary. Moreover, do not attempt to implement 'demes' and you may assume a single performance criterion for fitness (e.g. accuracy).
- Your instruction set should consist of the FOUR two argument arithmetic operators: $R[x] = R[x] < op > R[y]$ where $< op > \in \{+, -, \div 2, \times 2\}$
  - Note that you will need to consider how to protect operators or trap exceptions. See for example the discussion by [3].
- You should identify appropriate variation operators and assume a Breeder model as the selection–replacement operator (summarized below).
- You will also need to specify
  - the form assumed for your fitness function.
  - stop criterion.
  - the framework used to initialize your population of candidate programs.

Breeder model of selection–replacement

- Breeder model of selection–replacement defined as follows:

[1] A more detailed account is available in [2], however, this is only recommended as additional information.

1. Evaluate fitness, f_i, for all P individuals in the population.[2]
2. Rank the population and remove the worst $Gap\%$ individuals.
3. Apply variation operators to define $Gap\%$ offspring choosing the parents from the $1 - Gap\%$ individuals remaining in the population after step 2 with uniform probability.
4. Add the children to the population and return to Step 1.[3]

Multi-class classification

- Linear GP defines programs in terms of operations applied to registers (see above).
- Given a classification task consisting of C classes, we need at least C registers, $R[x] : x \in 0, ..., C-1$
- After program execution, the first C register references denote the confidence in labeling each class. Thus, $arg\max_x R[x]$ for $0 \leq x < C$ returns the GP individual's prediction of class label.

The assignment ask

- You will use two tasks for benchmarking your canonical GP classifier, specified as per Table 1. In each case 20% of the data should be reserved for test (i.e., not used for training your model) with exemplars selected to reflect the class distribution of the training partition.

  For example, if exemplars in a 4 class problem appear at a frequency of 50%, 25%, 10%, 15% across the dataset, then you want to partition your data into independent training and test partitions such that this distribution is maintained. Note that an exemplar can only appear in one of the partitions, *not* both!

  You should apply a standardization / rescaling to the data before attempting to construct GP models. Two commonly employed instances have the form:

  1. Mean and variance standardization

$$\acute{x}_i(p) = \alpha \times \frac{x_i(p) - \bar{x}_i}{\sigma_i^2} \tag{1}$$

  where $\bar{x}_i$ is the mean of attribute i over the training partition; σ_i^2 is the corresponding standard deviation; and α is a scaling parameter. A typical choice might be $\alpha = 5.0$. Note that the test partition would also require standardizing in the same way, making sure that you apply the same mean and SD as estimated from the training partition.

[2]Fitness implicitly assumes that larger values are better. Should minimization be necessary, then an appropriate normalization will be necessary.

[3]Actually only need to evaluate the fitness of the offspring, as the fitness of the parents has not changed.

Table 1: Source data for sandbox 1

Name	Data Source
Iris	`http://archive.ics.uci.edu/ml/datasets/Iris`
Tic-tac-toe endgame	`https://archive.ics.uci.edu/dataset/101/tic+tac+toe+endgame`

2. Linear transform or rescaling

$$\acute{x}_i(p) = \alpha \times \frac{x_i(p) - \min x_i}{\max x_i - \min x_i} \tag{2}$$

where $\max x_i$ is the maximum of attribute i over the training partition; and, $\min x_i$ is the minimum of attribute i over the training partition. Naturally, the *same* $\max x_i$ and $\min x_i$ values must be employed on the test partition as identified under the training partition. As per the case of Equ. (1) α represents a scaling factor, e.g. 10.0.

Reporting

- Write a short 4 page summary answering the following questions:
  - What was the fitness function and stop criterion you assumed? How did you go about initializing the population?
  - What parameterization did you assume for population size, maximum number of generations, application of variation operators and Gap?
  - Is there any advantage to letting the total number of registers exceed '*C*'?
  - Identify your best classifier, post-training, on the *training partition* and then report the percentage of each class this classifier identifies under the *test partition*. Why is such a policy assumed (as opposed to identifying the best model from the test partition)?
  - Characterize the performance of training using a plot of performance versus generations. Are particular classes identified[4] first, and other classes later? Is the same class identified first across multiple runs?
  - How did the number of instructions used by the champion at any generation develop over the generations? e.g. did it monotonically increase?
- You are free to choose a programming language for your implementation (e.g. Java, Python, R, or Matlab).
- Your report is due September 28 (submit on BrightSpace).

[4]By 'identified' we imply that a class is detected with $> 50\%$ accuracy.

References

[1] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, 2001.

[2] M. Brameier and W. Banzhaf. *Linear genetic programming*. Springer, 2007. `http://link.springer.com/book/10.1007\%2F978-0-387-31030-5`.

[3] J. Ni, R. H. Drieberg, and P. I. Rockett. The use of an analytic quotient operator in genetic programming. *IEEE Transactions on Evolutionary Computation*, 17(1):146–151, 2013.