# Report for CSCI6506 Sandbox 1

Jéssica Pauli de Castro Bonson (B00617515)

The implementation of the canonical GP for multi-class classification using a linear representation used the following final configurations:

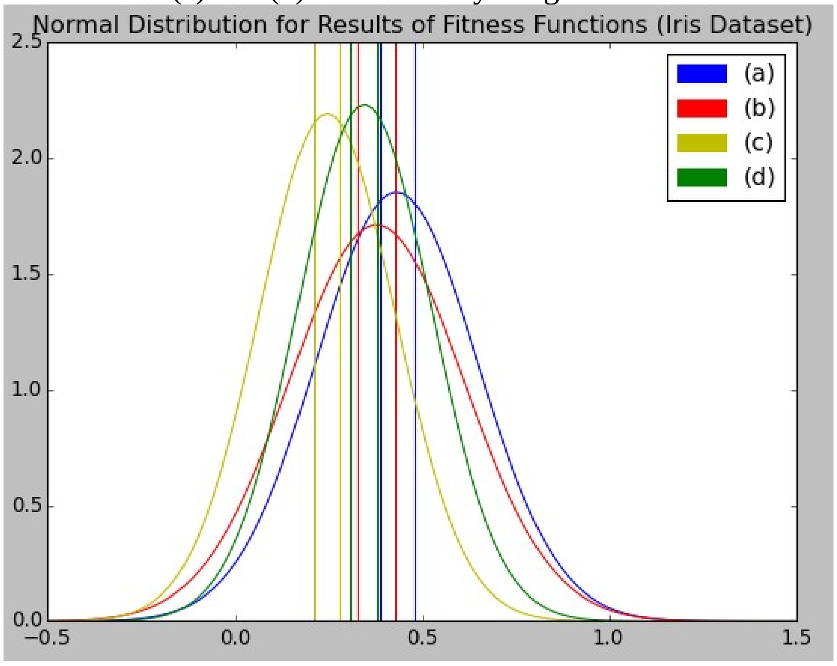
| **Parameters** | **Iris dataset** | **Tic-Tac-Toe dataset** |
|---|---|---|
| Population Size | 500 | 200 |
| Total Generations | 50 | 20 |
| Crossover Rate | 0.1 | 0.1 |
| Mutation Rate | 0.9 | 0.9 |
| Total Runs | 50 | 50 |
| Initial Program Size | 25 | 32 |
| Max. Program Size | 50 | 64 |
| Extra Registers | 2 | 2 |
| Fitness Function | $(\frac{1}{2}(MSEC + MSEI) + 2 * MA)/3$ | $MA$ |
| Selector Operator | Proportional Selection | |
| Variation Op. | Crossover and Mutation | |
| Stop Criterion | While current_generation < generation_total | |
| Best Result | 96.6% accuracy | 74.8% accuracy |

This values were obtained by comparing the result of sets of runs of the GP algorithm with each other, only modifying one variable per time. The comparisons were between the maximum, average and standard deviation accuracies between each set of runs. In the next paragraphs I will explain the details for some of these parameters, and then show the best trained programs for each dataset.

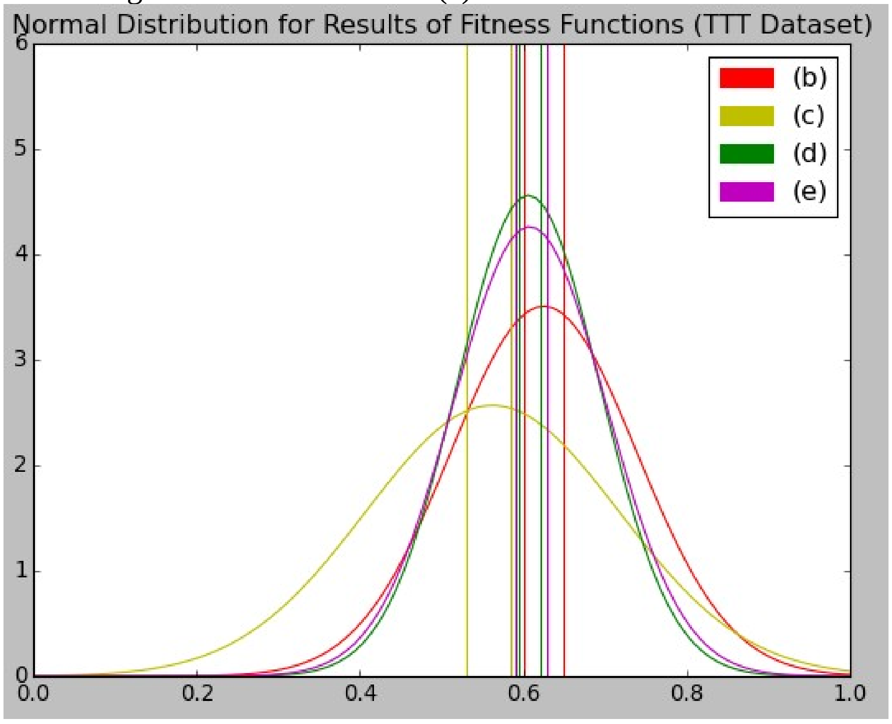
1. **Population Initialization:** The population is initialized randomly with an initial number of instructions equal to the Initial Program Size. The instructions structure follow the assignment specifications, using Protected Division (PD) and returning the first operand in case of overflow. I preferred to use PD over Analytic Quotient (AQ) since it considerably skewed the data for small numbers, common in both datasets.

2. **Fitness Function:** For the Iris dataset I tried four types of fitness functions. I will explain the best one I found first and then briefly talk about the other ones. The raw equation is:
$$(\frac{1}{n*(1+m)}(\sum_{i=1}^{n}(\widehat{A}_i - 1.0)^2 + \sum_{i=1}^{n}\sum_{j=1}^{m}(\widehat{B}_i - 0.0)^2) + 2 * \frac{1}{k}\sum_{i=1}^{k} acc_i)/3$$
Where A are the results from the membership function for the output that was predicted as the correct one, and B are the results for the classes predicted as incorrect. *n* is the sample size, and *m* is the size of incorrect classes. So the first part of the equation is the total average of the mean square error (MSE) for the correct (C) and the incorrect (I) classes. The second part of the equation is the mean of the total accuracies (MA). The weight value (equal to 2) just ensures that the fitness function rewards more a correct result than a well separated one. So the short version of the fitness function is:
$$(\frac{1}{2}(MSEC + MSEI) + 2 * MA)/3$$
This fitness function (a) produced 96.6% of max. accuracy. The second best fitness function (b) was using just the MA part of the equation, with a max. accuracy of 90%. It is worth

noting that this one was very prone to overfitting. The other two fitness functions both obtained 66.6% as their maximum accuracy. The first one (c) didn't used MSEI in the equation, just MSEC and MA. The second one (d) instead of the MSE, used the difference between the membership values of best class output versus the second best class output. The next figure shows the data distribution of the results for fitness functions (a), (b), (c) and (d) with a confidence interval of 0.85. By this distribution we can see, at a 0.15 level, that the hypothesis of (c) and (d) being better than (a) can be discarded, that (c) is definitively the worst hypothesis, and that, besides (a) having provided a final best solution, the difference between (a) and (b) is statistically insignificant.

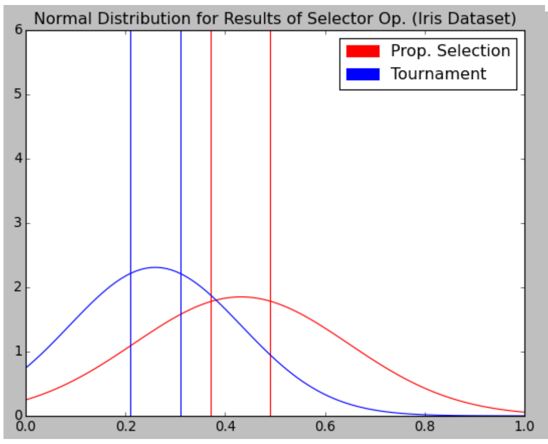


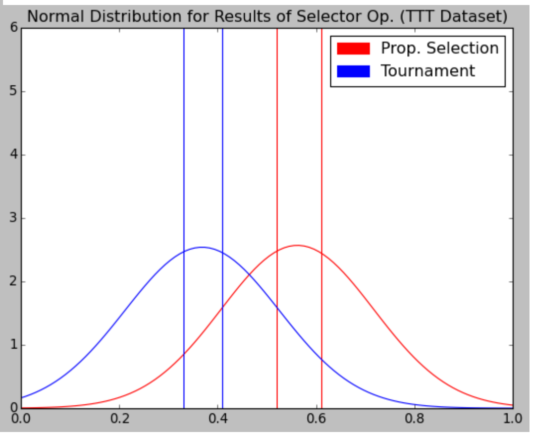
For the Tic-Tac-Toe dataset I tried the same 4 fitness functions and an additional (e) function based on MA that increases the fitness the more inputs are being used, going from 0.0 to 0.9 (0.1 for each type of input). The following figure shows the data distribution for the results of the fitness functions (b), (c), (d) and (e) with a confidence interval of 0.85. (a) will not be shown because it got the same interval as (c).



By the distribution we can see that (c) was definitely the worst one, while the other three had very statistical similar results, with a bit better results for (b), that is the one who produced the best solution.

3. **Variation Operators:** I decided to support the crossover and the mutation operators, since crossover allows a continuous evolution of the population, and mutation enables the algorithm to insert new data. The crossover randomly gets a set of instructions from a parent and exchanges it with a random set from the second parent. In case the offspring has more than the maximum number of instructions, the last ones are removed. The mutation randomly modifies either the mode, target, operand, or source of a random instruction, ensuring that the produced instruction uses registers in a valid range. The replacement occurs by replacing the worst individuals in the population/tournament by the offspring.

4. **Selection Operators:** To be able to compare the results from the Proportional Selection with the Steady Tournament, it was necessary to ensure that they performed an equivalent number of evaluations. The parameters for the Isis dataset resulted in 25000 evaluations, which is equivalent to 6250 generations of tournaments with 4 participants each. For the Tic-Tac-Toe dataset, there was 4000 evaluations, and thus 1000 tournaments. The figure below shows the data distribution of the results for 50 runs with a 0.05 level.




Its clear by the data distribution that the Proportional Selection outperformed the Steady Tournament in both datasets. By analyzing the training and test error we can see that what happened in that the Steady Tournament heavily overfitted for this amount of generations. Besides the distribution results, the main advantage I noticed for the Proportional Selection over the Steady Tournament was that it was easier to understand the output and tune the algorithm parameters.

5. **Best Classifier:** The best classifier codes are bellow. The best classifier for the Iris dataset obtained an accuracy in the test dataset of 96.6%, with 90.9% accuracy for the first class (Iris-setosa) and 100% accuracy in the other classes (Iris-versicolor and Iris-virginica). At the end its fitness was 0.6625 and its accuracy in the training set was 88.3%.  The final version of the solution was generated at the 50th generation.  The best solution for the Tic-Tac-Toe data obtained  a accuracy of 74.8% in the test dataset (81.7% for the positive class, 61.9% for the negative). It was produced in the 11th generation, with a fitness of 0.689 and an accuracy in the training data of 68.9%.
It is relevant to note the Isis solution was more complex, since there seems to be to introns and it has 41 instructions, while the Tic-Tac-Toe solution has 30 instructions, with 8 introns. Probably tuning the parameters for the Tic-Tac-Toe algorithm so it can produce more complex solutions would give a better accuracy.
The policy of training and testing the programs in different datasets occurs because this way you can test how well the solution generalizes for new data, avoiding solutions that just performed good on the training data because they overfitted.

| | |
|---|---|
| r[3] = r[3] - i[1] | r[1] = r[1] + r[0] |
| r[0] = r[0] - i[0] | r[1] = r[1] - i[6] |
| r[2] = r[2] + r[0] | r[3] = r[3] * i[5] |
| r[3] = r[3] - i[0] | r[1] = r[1] / r[1] |
| r[3] = r[3] - r[1] | r[3] = r[3] * i[7] |
| r[1] = r[1] / r[1] | r[2] = r[2] / r[2] |
| r[2] = r[2] + r[0] | r[2] = r[2] * i[6] |
| r[3] = r[3] / r[1] | r[2] = r[2] / i[3] |
| r[2] = r[2] - r[3] | r[1] = r[1] - i[2] |
| r[2] = r[2] / i[1] | r[3] = r[3] * i[0] |
| r[1] = r[1] + i[1] | r[0] = r[0] + i[8] |
| r[2] = r[2] - i[3] | r[1] = r[1] * i[7] |
| r[0] = r[0] * i[0] | r[1] = r[1] / i[4] |
| r[0] = r[0] * r[0] | r[0] = r[0] - r[3] |
| r[3] = r[3] / i[3] | r[1] = r[1] + i[4] |
| r[4] = r[4] + i[0] | r[1] = r[1] * i[2] |
| r[2] = r[2] * i[0] | r[3] = r[3] / i[1] |
| r[2] = r[2] - i[3] | r[0] = r[0] * r[2] |
| r[2] = r[2] - i[0] | r[2] = r[2] + r[2] |
| r[2] = r[2] * i[0] | r[2] = r[2] - i[5] |
| r[1] = r[1] + r[1] | r[2] = r[2] / r[1] |
| r[3] = r[3] / i[3] | r[1] = r[1] / r[0] |
| r[4] = r[4] / i[0] | r[2] = r[2] - i[1] |
| r[3] = r[3] - r[2] | r[2] = r[2] * i[0] |
| r[4] = r[4] / r[1] | r[2] = r[2] * r[0] |
| r[2] = r[2] - i[1] | r[0] = r[0] * i[8] |
| r[3] = r[3] / i[0] | r[3] = r[3] - i[6] |
| r[3] = r[3] - i[1] | r[0] = r[0] + i[4] |
| r[0] = r[0] / r[1] | r[0] = r[0] + i[4] |
| r[3] = r[3] * r[2] | r[0] = r[0] * i[4] |
| r[0] = r[0] - i[2] | |
| r[2] = r[2] / i[1] | |
| r[3] = r[3] * i[0] | |
| r[0] = r[0] / i[3] | |
| r[0] = r[0] * i[1] | |
| r[0] = r[0] + r[3] | |
| r[1] = r[1] / i[3] | |
| r[0] = r[0] / r[1] | |
| r[1] = r[1] / r[4] | |
| r[2] = r[2] * r[3] | |
| r[2] = r[2] / r[1] | |
| ***Code for best program (Iris dataset)*** | ***Code for best program (Tic-Tac-Toe dataset)*** |