Student Name: ***Sebastian Good***

Student ID: ***14002050***

Degree: ***Robotics***

Module Code: ***UFCFY3-15-3***

Title: ***Bio-Computation: Assignment***

Contents

INTRODUCTION **1**

BACKGROUND RESEARCH **2**

USES OF DATA CLASSIFICATION AND DATA MINING 2

Extracting knowledge from a text database *2*

Market Rules from User Data *2*

Fraud Detection *2*

EXPERIMENTATION **3**

THE GENETIC ALGORITHM 3

DATASET 1 3

Dataset 1: Fitness Graph *4*

Dataset 1: One-tenth population size. *5*

Dataset 1: Ten times population size. *6*

Dataset 1: Triple Mutation Rate *7*

DATASET 2 7

Dataset 2: Fitness and Length Graph *8*

DATA SET 3 9

Dataset 3: Fitness and Length Graph *9*

Dataset 3: Training Fitness vs. Full Fitness *10*

CONCLUSIONS **12**

REFERENCES **13**

Introduction

In this assignment, this report was given the task of attempting to classify three sets of data provided to us using any form of evolutionary intelligence covered in the bio computation course. These datasets were provided in text files (.txt) from Blackboard. This report was given many methods of classification including; look-up tables, rule-based systems and neural networks. For the given task, it was decided to build upon the genetic algorithm code developed in the first few lab sessions, which means a 'rule-based' system was chosen as the avenue of investigation.

The first two assigned datasets were 'simple' binary, the third dataset was a set of floating point number. The first data set was thirty-two rows, long containing 5 binary input variables and 1 output variable. The second set was sixty-four rows long, containing 6 binary input variables and a single output variable. The final dataset was two-thousand rows long and contained 6 floating point variables and a single binary output variable.

The genetic algorithm this report uses will attempt to search the 'problem space' by attempting to widely populate the problem space, then through a process of tournament selection, ruleset crossover and random mutation and then judging their world model. Then continuing this process, to narrow down the problem space's best solutions until the process is either stopped manually or a success threshold has been reached.

This report uses the Genetic Algorithm to create a rough approximation of the environment which the data is in, producing rules that predict whether the dataset will be characterized by a one or a zero.

Background Research

The act of data classification or data mining is common for many businesses. The modern world produces Terabytes of data every second, and it would be too time consuming to try and search through it all for patterns by hand, its far easier to get a program to attempt to identify the patterns for the business, especially if the patterns are subtle or hidden. A program that can take in data with predefined given outputs and produce a set of rules that could be used to classify new and unknown data.

Uses of data classification and data mining

Below this report has listed common use cases for data classification and data mining.

Extracting knowledge from a text database

This is the extracting of data from a text document to help categories its content. For this type, the report is decoded and key dictionary terms are selected, before being compared to a database of concepts relating to given dictionaries. From this, the content is estimated. This is useful for customer service emails to help an organization pre-sort the emails, before they're given to customer relations departments. (Sakurai, 2001)

Market Rules from User Data

For this, data about the customers is fed into the program then the attempts to create rules of association for that database. So, a shop may use its club card to take data such as, age, where the customer lives, gender and job. Then the program correlates the user's information with the users buying patterns, so that the system can try to work out what products to place together in a store, or what offers would apply to the customer to convince them to spend more in-store. The program is thus trying to ascertain purchasing probability based on all current data collected of the customer. (Terano, Ishino, 1996)

Fraud Detection

For this, the system is feed a large array of customer purchase data, and it's tasked with spotting fraud. For this, the system is constantly building and testing the fitness of the rules. And with the testing becoming accurate after a certain point. If a huge change in fitness

occurs it then goes back and tries to find the variable that caused it, and that is the fraud. (Bentley, 2000)

Experimentation

Below is a detailed explanation of how this report approaches the proposed problem.

The Genetic Algorithm

With the three data sets, this report deemed the best approach would be to attempt to consider them as generically the same and hence create a program that can accept and manipulate the datasets with minimal changes to that program. For this reason, JavaScript was selected as JavaScript does not discriminate between different variable types, instead it has a single variable type; ‘var’. As such, if the core code works for the first data set it should, in theory work for all the proceeding data sets with only a few minor changes to calling function. The main program components to change would be how the rules are tested, and the precision of inputs.

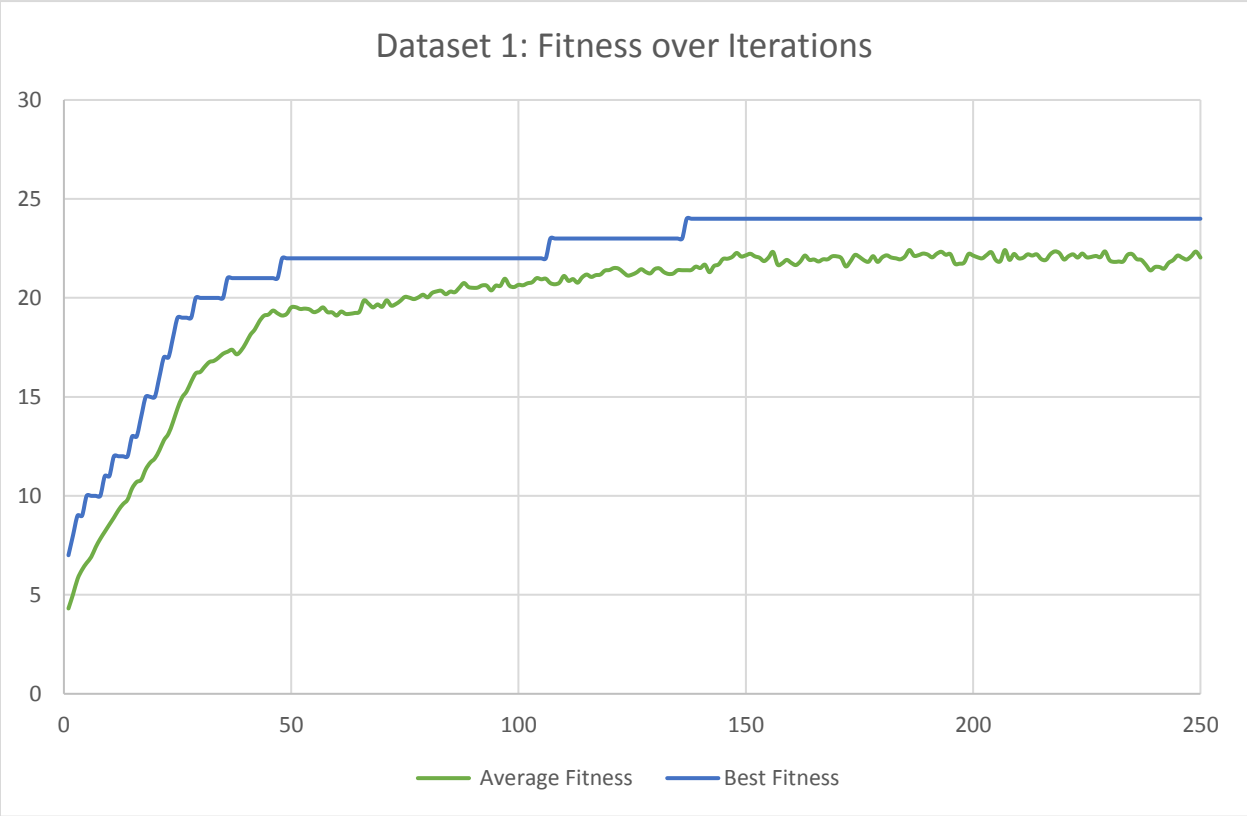
To test the fitness of the rulesets, each set of rules would be compared with the training data provided to see how many data points match the rules. The rules will be composed of ‘normal’ numbers and ‘wildcards’, ‘wildcards’ will serve the purpose of saying a data point is not needed for an outcome. The other main design point is that the algorithm will be able to increase or decrease the number of rules needed to attempt to classify the data set.

The genetic algorithm works using four basic methods and a couple of fundamental ideas. The four methods are as follows; Evaluation of rule sets, Selection of generally better rules, Mutation of random rules at random genes and Crossover of two or more randomly selected pairs. The fundamental ideas consist of always using floating point numbers, (even for binary datasets) then rounding them to the correct precision and having a modular code base to ensure the base genetic algorithm should never change.

Dataset 1

This is the smallest data set containing 32 rows of data points, each with their own output number. This could be considered the practice or proof of concept set dataset. The main part to this data set is testing to see if the basic rule classification works. This ensures the program's basic functionality and could be used to ensure no unexpected results are achieved.

Dataset 1: Fitness Graph



As the above graph, shows the initial rapid increase of fitness is followed by a leveling out as the genetic algorithm approaches the highest possible fitness. The maximum fitness that this genetic algorithm achieved for the first data set is twenty-six, the following is the ruleset that achieved this: The rulesets were parsed into the following data structure:

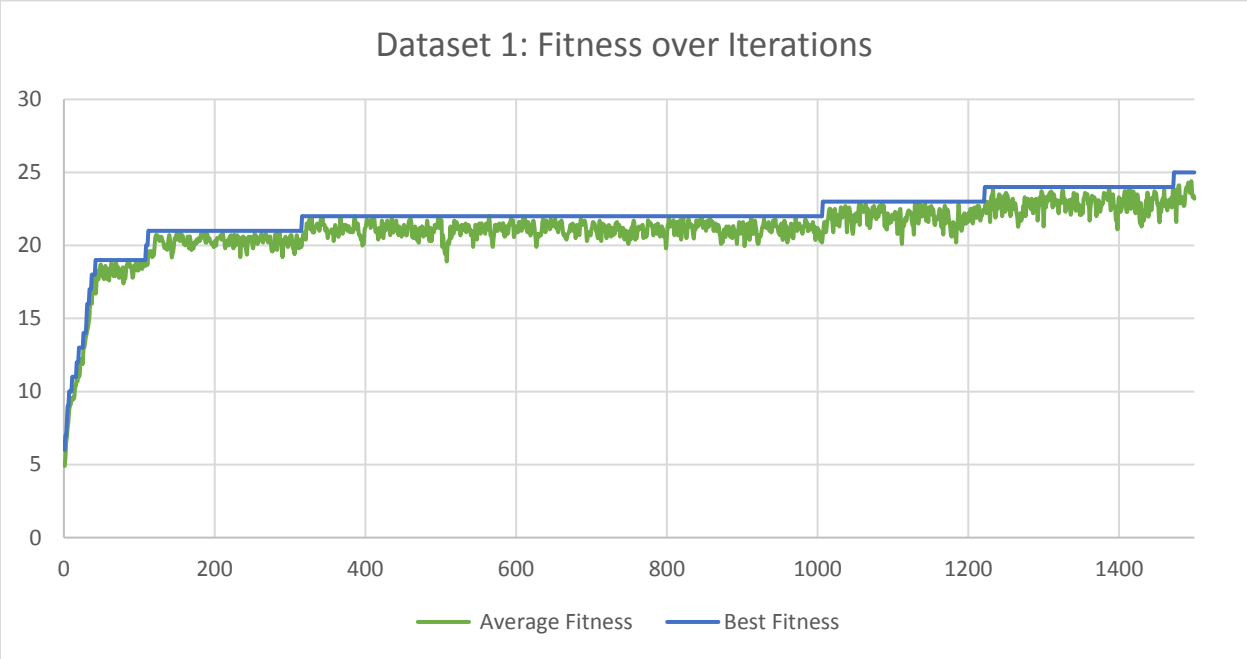
```
#, #, 0, #, 0, 0,
0, 1, 1, 1, 1, 1,
1, 1, 1, 0, 1, 1,
#, #, 1, #, 1, 0,
0, 0, #, 1, #, 1,
```

```
{
    input: [
        integer,
        integer,
        integer,
```

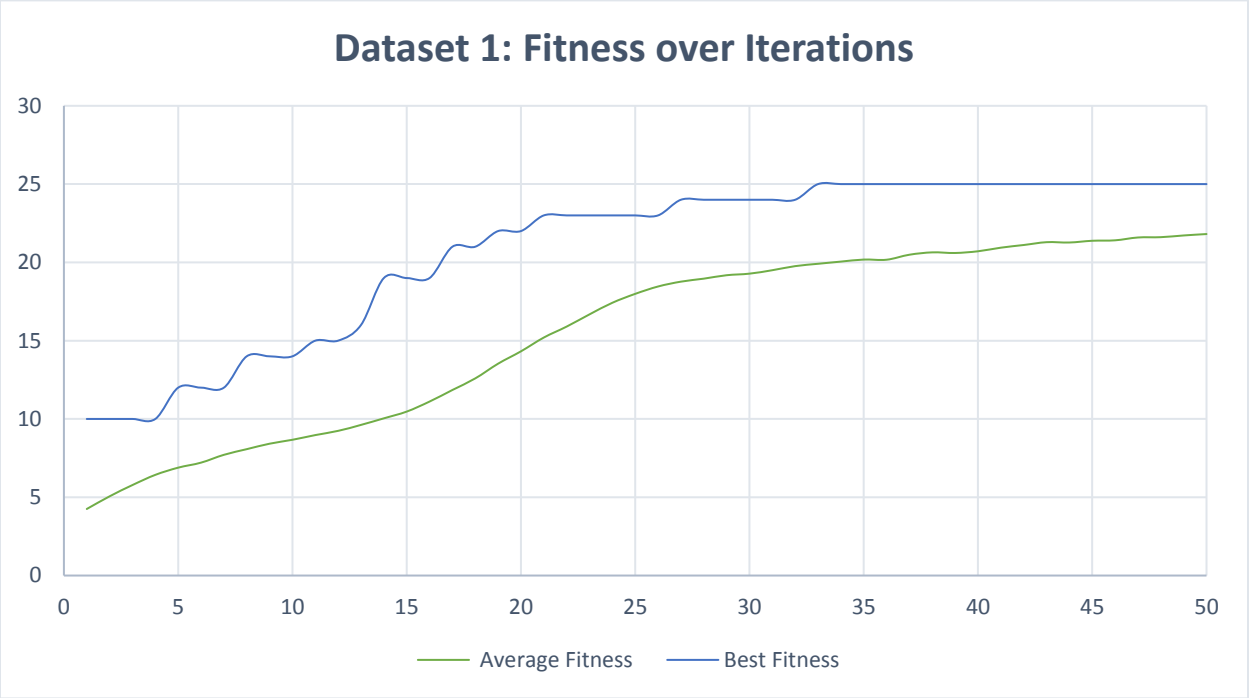
```
0, 1, #, 0, #, 1,                    integer,
1, 0, #, 0, #, 1,                    integer
1, 1, 0, 1, 0, 0,                 ],
1, 1, #, 1, #, 1,                 output: [integer]
#, #, #, #, #, 0              }
```

Unfortunately, the full data run that achieved this was not recorded as it was only achieved in a testing phase, however it was recorded as it was the highest achieved up to that point.

Dataset 1: One-tenth population size.



Dataset 1: Ten times population size.



The graphs above demonstrate how increasing and decreasing population size dramatically changes the number of iterations, the larger the population the fewer iterations that are required to classify the data.

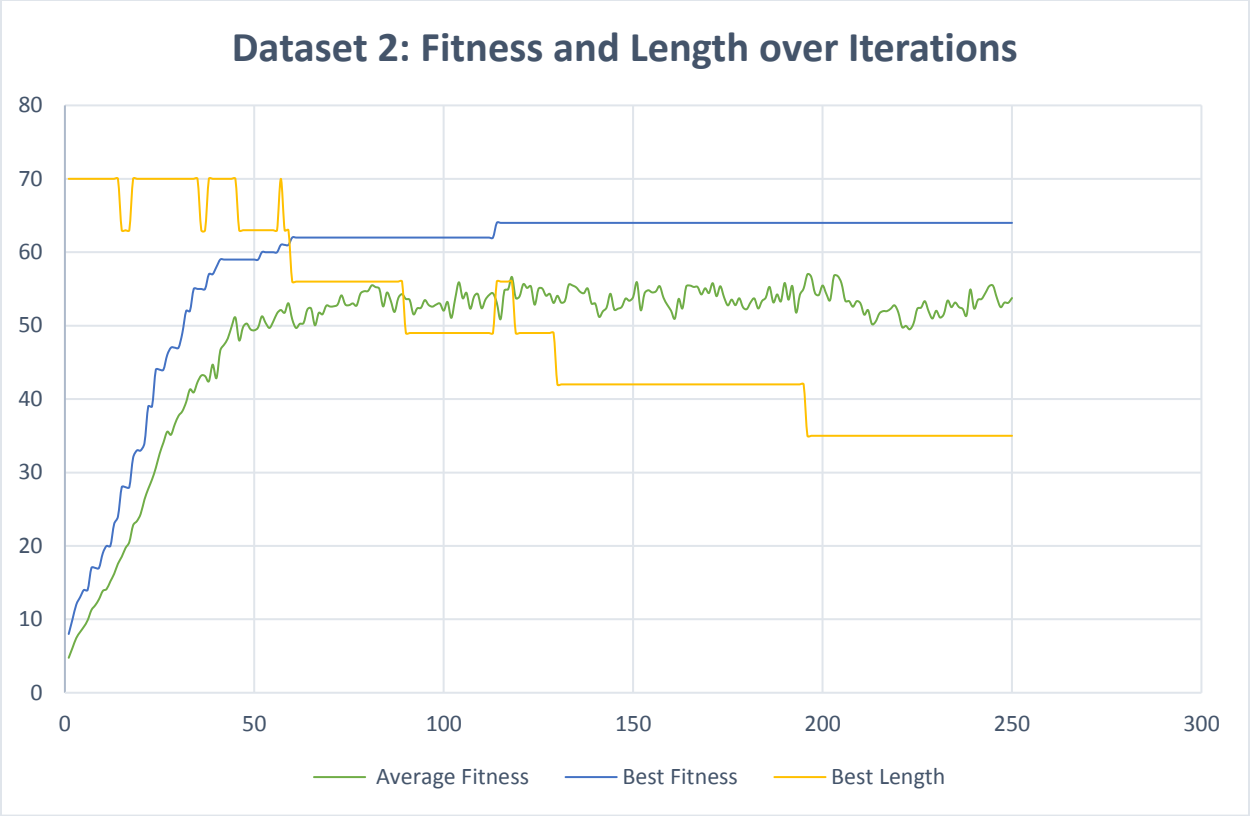
Dataset 1: Triple Mutation Rate

This graph demonstrates how the triple mutation rate keeps the average fitness down due to the gene sequences not being able to settle due to the better sequences being mutated to a worse fitness state.

Dataset 2

The second dataset is double the length of the first dataset and contains a relatively easy to identify pattern, the main aim for this dataset is testing whether the genome length mutation worked. This allowed the program to decrease the number of rules used in comparing against the dataset, while keeping the average fitness the same, with the goal of getting the fewest rules to classify the data. This was achieved by including in the 'selection' process a method saying, 'if the fitnesses are the same, select the shortest one' and then including in the 'mutation' process a 'either generate new rule and add it to the sequence or remove a rule from the current sequence'. This dataset was parsed into a data structure similar to the structure used for dataset one, however, this time there were six input integers as the data had six input variables.

Dataset 2: Fitness and Length Graph



The graph above shows how over time after a relative maximum fitness is reached the length of the gene sequences decreases. At one-hundred and thirteen iterations the length increases again where a new best sequence is found, this is then shortened again where a rule that was once relevant became redundant. The best result achieve by this genetic algorithm was a fitness of sixty-four consisting of five rules:

```
0, 0, #, #, #, 0, 0,
0, 1, #, #, 0, #, 0,
1, 1, 0, #, #, #, 0,
1, 0, #, 0, #, #, 0,
#, #, #, #, #, #, 1
```

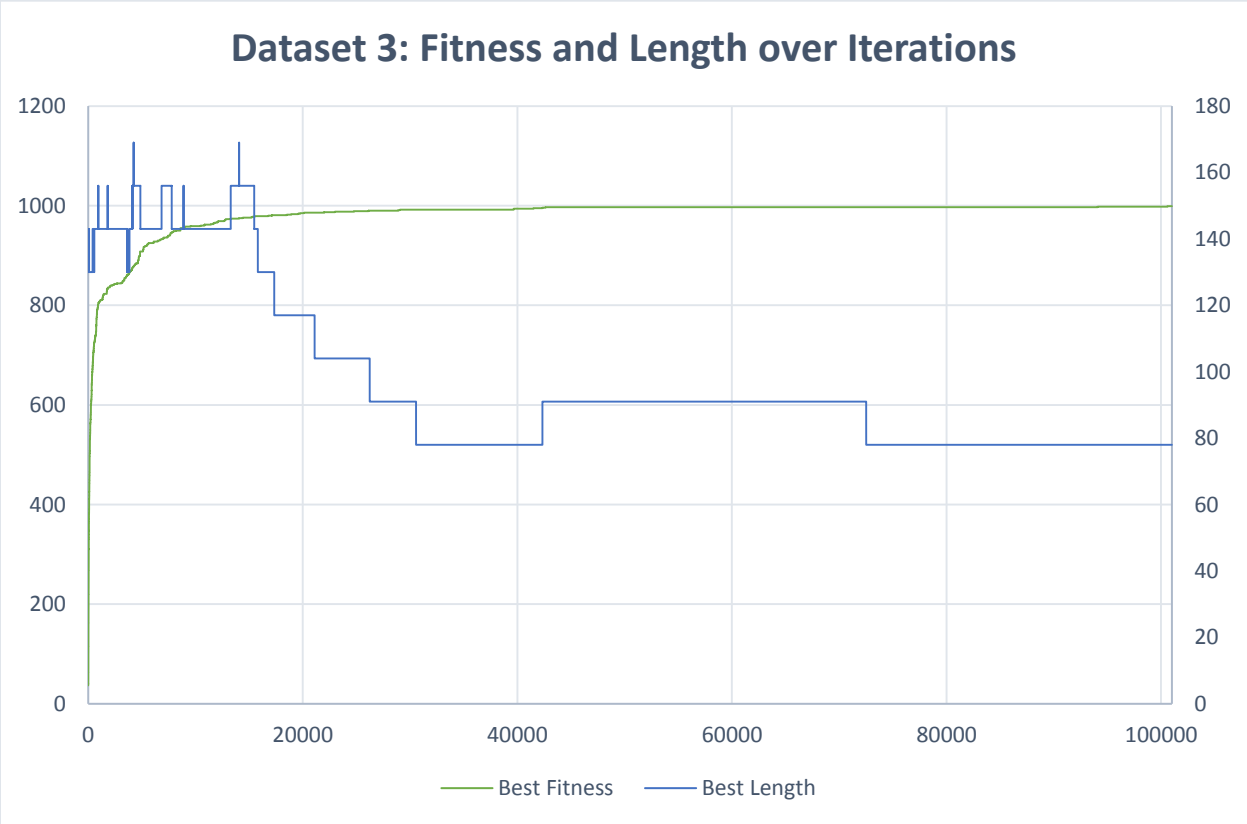
It is clear to see from the best rules produced by the genetic algorithm that it successfully discovered a pattern in the data and build the rules accordingly.

Data Set 3

This was the most complicated dataset as it contains floating numbers and a different formatting, in the input file. However, using JavaScript, the main change was in the rule comparison, as the comparison method needed to compare between 2 rule points. This also means the rule's input length would need to be increased to allow for both floating point 'bounds'. The new structure that each rule will take after being parsed will be as follows:

```
{
    input: [
        [float, float], [float, float], [float, float],
        [float, float], [float, float], [float, float]
     ],
    output: [integer]
}
```

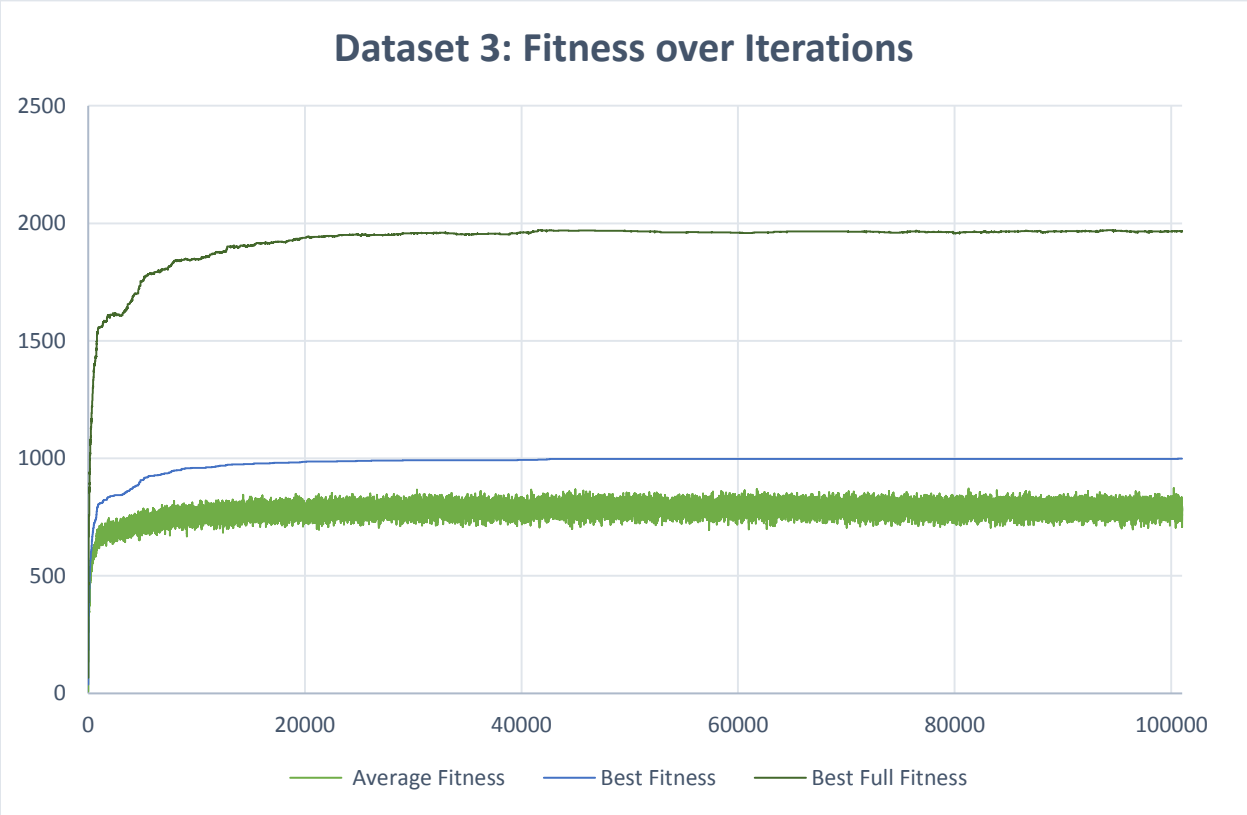
Dataset 3: Fitness and Length Graph



The above graph demonstrates the characteristic dramatic initial fitness increase and then smooth levelling out of the fitness as the length then procced to decrease. As the genetic algorithm was only presented with the training dataset it was necessary to test the best data

sequence against the full dataset. It was decided to check every iteration to produce a comprehensive result even though this was not required and would slow the data analytics down. It would be more appropriate to test against the full dataset every 'n' iterations if speed was preferred. The training data set was the first one-thousand rows of the full dataset, this could be increased or decreased depending on the user's preference.

Dataset 3: Training Fitness vs. Full Fitness



The above graph demonstrates that the training data was a very good match as there doesn't appear to be any visible over fitting of the training data compared to the full dataset.

The best result achieved for the third dataset was a fitness of one-thousand nine-hundred and thirty of length seventy-eight (six rules) when they're applied to the full data set. The rules set was as follows:

```
[{
    "input": [
        [0.009833, 0.496857], [0.475597, 0.994618],
        [0.005272, 0.968161], [0.000692, 0.500729],
```

```
        [0.002018, 0.991143], [0.000206, 0.994653]
    ],
    "output": [0]
}, {
    "input": [
        [0.001789, 0.485546], [0.006215, 0.496942],
        [0.002730, 0.498536], [0.002821, 0.997299],
        [0.026874, 0.998409], [0.004486, 0.996405]
    ],
    output": [0]
}, {
    "input": [
        [0.468979, 0.607131], [0.699226, 0.807686],
        [0.020254, 0.804398], [0.233470, 0.816951],
        [0.106258, 0.447094], [0.622610, 0.863317]
    ],
    output": [0]
}, {
    "input": [
        [0.488461, 0.999185], [0.493616, 0.998730],
        [0.005810, 0.998991], [0.000563, 0.987334],
        [0.019247, 0.997774], [0.002508, 0.498341]
    ],
    output": [0]
}, {
    "input": [
        [0.498656, 0.994997], [0.002204, 0.491880],
        [0.003967, 0.993339], [0.000346, 0.987049],
        [0.002722, 0.503912], [0.030871, 0.996704]
    ],
    output": [0]
}, {
    "input": [
        [0.001013, 0.999727], [0.001105, 0.998219],
        [0.002183, 0.999708], [0.005913, 0.999691],
        [0.002795, 0.997870], [0.003140, 0.998086]
    ],
    "output": [1]
}]
```

Conclusions

The experimentation demonstrated the general principles of genetic algorithms and how they settle over time, along with the randomly decreasing or increasing ruleset length, showing how genetic algorithms traverse their search space. The datasets supplied, provide excellent examples of how simple manipulations in a properly planned program can easily compensate for the differences in numerical datasets.

Datasets one and two were useful in demonstrating the core functionality and brought a few flaws in this implementation of a genetic algorithm to the fore, such as the initial process of mutation and crossover, where crossover points were selected at random and mutation could strip or add rules from and in incorrect locations. The random point selection worked wonderfully with rulesets of the same length, however upon ruleset length mutation the parsing of rules became broken as some parts of rules were lost. The random length changes were initially only removing and adding rules to the end of the set, however, this was changed to allow for rule addition and subtraction in any rule location within the ruleset. This gave the advantage of a more efficient fitness increase.

Data set three presented a much more interesting effect as wildcards became redundant. This was because the dataset inputs were floating point, requiring solely floating point bounds.

In theory, the genetic algorithm if provided solely with a ‘training dataset’ could overspecialise in the training, causing the fitness of the ‘full dataset’ to decrease. This was not seen in this report’s classification trials, however, it was a concern that was brought to light.

If this report was to be revisited, an alternative method of data classification that would be most likely to be investigated, is the use of neural networks. The reason for this is the fact that given a suitable sized dataset, a neural network could build a model that could be trained as new data arrives, unlike the genetic algorithm that most likely would need to be restarted whenever new data is provided.

References

Sakurai, S., Ichimura, Y., Suyama, A., & Orihara, R. (2001). Acquisition of a knowledge dictionary for a text mining system using an inductive learning method. IJCAI 2001 Workshop on Text Learning: Beyond Supervision

Terano, T., & Ishino, Y. (1996). Knowledge acquisition from questionnaire data using simulated breeding and inductive learning methods. Expert Systems with Applications

Bentley, P. J. (2000). "Evolutionary, my dear watson" investigating committee-based Evolution of fuzzy rules for the detection of suspicious insurance claims. Genetic and Evolutionary Computation Conf. (GECCO-2000). Morgan Kaufmann.