**1 INTRODUCTION**

This assignment consisted of writing an implementation of Genetic Algorithms (GA) in order to breed a useful rule based data classifier for 3 seperate sets of data. Data Sets 1 and 2 using binary strings with equals comparison, and Data Set 3 using a list of floats with a range based comparison.

The aim would be for the implementation to find structure in the data given and generate individuals with rule sets that can classify data based on this.

The code used in this project can be found via a link in the appendix as figure 3

**2 BACKGROUND RESEARCH**

Data mining is the process of using a set of variables in order to come to a conclusion about something. “Data mining is most commonly used in attempts to induce association rules from transaction data.” (Tzung-Pei Hong et al.) Banks can often use past data on loaners, taking into account their income, postcode, number of dependants etc and their return rate as a way to build a classifier to determine is a particular individual is a borrower that is likely to pay their loan back.

Verma, G and Verma, V also discuss how data mining is useful for businesses with the scenario of how “analysis of retail point of sale transaction data can yield information on which products are selling and when”. Although it is not a problem that this report has to deal with “feature selection is an essential problem for pattern classification.” (MaryamYassi et al.) The successful classification rate can in turn be used to determine if a particular set of features can be used to determine a particular outcome, with high classification rates being particularly good at predicting an outcome.

As the amount of data on people in particular has risen, “vigorous efforts have been devoted to designing efficient mechanisms for mining information and knowledge from large databases.“ (Hong, T., Lee et al.)

Genetic Algorithms “are used to generate useful solutions to optimization and search problems” (Verma, G. and Verma, V) In the case of data mining, the input data can be considered the search space, with the outputs from the test data being used as a baseline to optimize towards.

However, data classifiers are used for more than targeted advertising and claim checks. A particular study was done by Shital Shah et al. where “a training data set with the 167 genes” was used to evolve a data classifier to find cases of ovarian cancer. The data in this scenario were made from the genetic data of actual people. The results of this study gave a classifier with a 97.78% success rate for finding people with cancer, and a 93.38% success rate for finding people without cancer.

**3 EXPERIMENTATION**

Each data set makes use of the same simple genetic algorithm, where N, M and the specific fitness function are dependant on the data set used:

Create population of 500 individuals w/ N rules  
Evaluate initial population  
While max fitness is not found in population  
 Generation limit (1000) is not reached:  
 Select parents using tournament selection  
 Create offspring by crossover of parents   
 with 100% chance  
 Mutate offspring with M chance  
 Evaluate offspring using specific fitness function  
 Replace worst in offspring with best in pop  
 Replace population with offspring

Rather than have to also test with different rule amounts in order to find effective small solutions, the number of rules that are used in evaluation are also mutable values, generated between N and N/2.  
  
To prevent the program from running for a potentially infinite time at particular settings the Generation limit was set to 1000 generations. After this point it would stop and print the current highest fitness individual.

**3.1 Data Set 1**

Data Set 1 consists of 32 examples with a 5 bit input and a 1 bit output, making for a total of 6 bits. The individuals are generated with 10 rules, making for a genome length of 60 bits, which can be split into individual rules at time of evaluation.

Wildcards '#' were mixed into the possible values for each input during initial generation and mutation, which count as a success when compared against a bit from an example regardless of that example bits value. Without this an individual wouldn't be able to classify more pieces of data than it has rules. Wildcards allow for a small set of rules to classify larger sets of data.

The fitness function used for Data Set 1 was short -

For each example, iterate over each rule in individual  
 If example input matches rule input  
 If example output matches rule output

Increase individual's fitness by 1

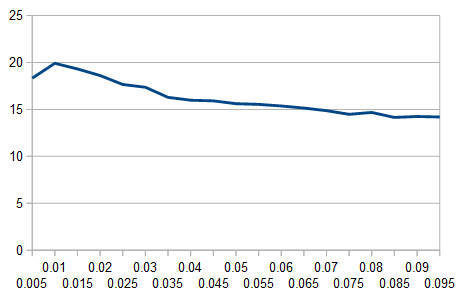
Stop comparing, move to next example

Else

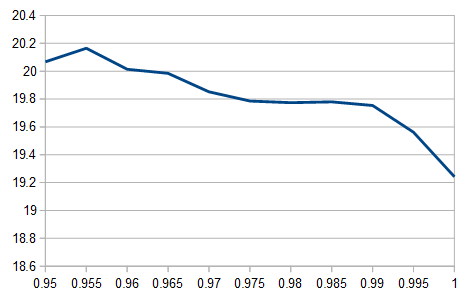
Stop comparing, move to next example

To get a good idea of suitable parameters I ran a parameter sweep over mutation rate from 0.5% to 10% with 0.5% steps, then a sweep over crossover using the best mutation rate between 95% and 100% with steps of 0.5%. The mean value of each run was then plotted against the mutation and crossover rates to give these results.

Mutation against mean fitness



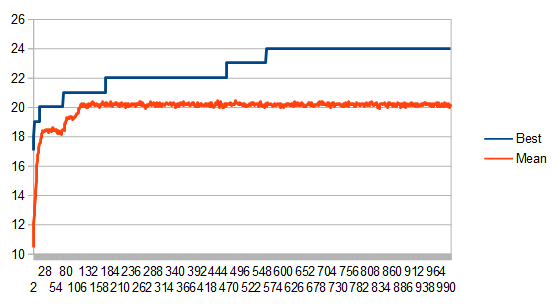
Crossover against mean fitness



Here we can see that a crossover of 1% works best, likely due to a higher mutation rate not allowing for good individuals to spread through the population before being changed, and a lower mutation rate not giving enough variation in individuals.

From these results I can determine that a mutation of

1% and a crossover of 95.5% is most suitable. Running 10 runs against these parameters and taking the best run gives can allow us to see how a strong population evolves and give us a strong individual, with a fitness of 24.



Best Individual - 24 Fitness, 8 Rules

1, 0, 0, 1, 0, 1

1, 0, 1, 1, 1, 1

0, 1, 1, 1, 1, 1

0, 0, 1, 1, 0, 1

1, 1, 1, 1, 0, 1

1, 1, 0, 1, 1, 1

'#', '#', '#', '#', '#', 0

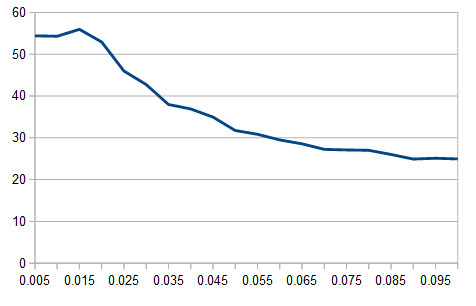
'#', 1, 0, 0, '#', 1

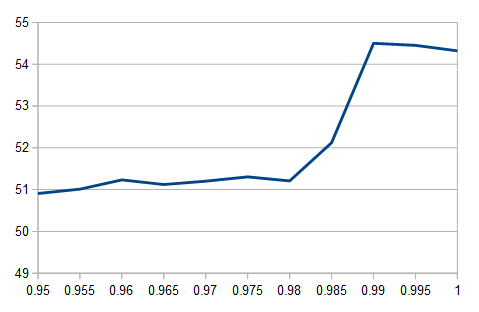
**3.2 Data Set 2**

Data Set 1 consists of 64 examples with a 6 bit input and a 1 bit output, making for a total of 7 bits. The individuals are generated with 10 rules, making for a genome length of 70 bits.

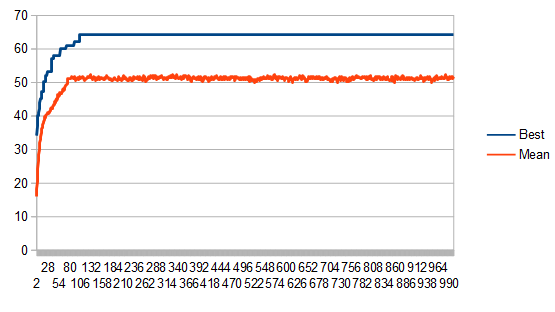
The fitness function remains the same as Data Set 1 due to them both being a list of binary inputs and single output.

Again a parameter sweep was done with the same bounds as data set 1 and plotted on graphs, mutation of 0.5% to 10%, Crossover of 95% to 100%.

Mutation against mean fitness

Crossover against mean fitness

Unlike Data Set 1, all 10 runs of 99% Crossover and 1.5% Mutation managed to finish with a 64 fitness individual with 5 rules. One of them is shown below, which reaches max fitness very quickly.



Best Individual – 64 Fitness, 5 Rules:

0, 1, '#', '#', 0, '#', 0

1, 0, '#', 0, '#', '#', 0

1, 1, 0, '#', '#', '#', 0

0, 0, '#', '#', '#', 0, 0

'#', '#', '#', '#', '#', '#', 1

**3.3 Data Set 3 (and UCI data)**

Data Set 3 consists of 2000 examples with a 6 float input and a 1 float output, making for a total of 7 floats. The individuals are generated with 10 rules, making for a genome length of 70\*2 floats. The number of floats in a genome is doubled to account for the change to the fitness function shown below.

FOR each example, iterate over rule in individual

FOR N example input in example

IF N example input > N\*2 rule input

IF N example input < (N\*2)+1 rule input

IF example output = rule output

fitness += 1

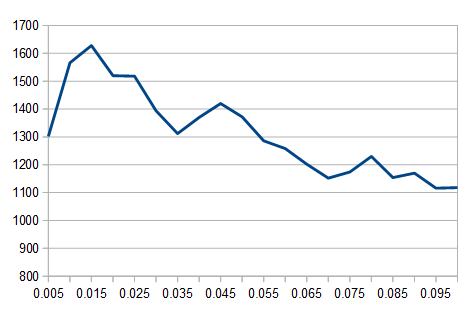
ELSE Skip to next example

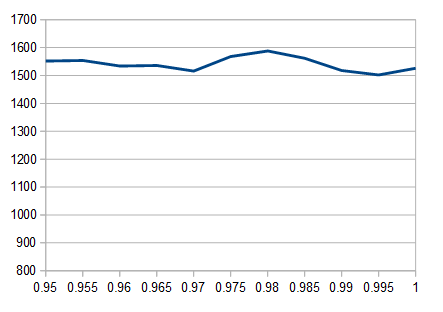
Rather than do direct comparisons, each float in an example is compared to be inbetween a pair of floats in an individual. (10 example inputs, 20 rule inputs)

Mutation of the individual also changes, +/- 0.1, but still bound between 0.0 and 1.0. This allows for pairs of inputs, lower of 0.0 or higher of 1.0 to emulate the Wildcard scenarios in the binary data set methods.

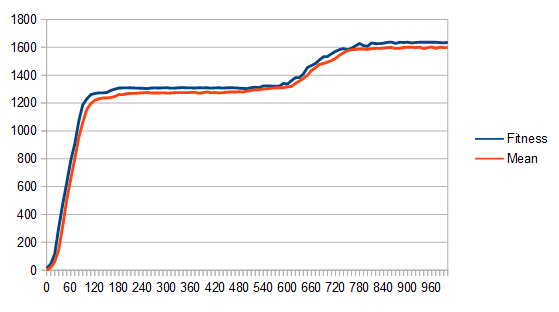
To adjust for being given a large data set, only 1000 of the examples were compared against when evaluating, except on every 10th generation in which the entire data set was compared against. This both speeds up the execution and allows for the use of the first 1000 as a training set, to which the full evaluation can be used to measure overfit. The population is also increased to 1000, twice as high, due to the larger data to map.

Again, sweeps were done between 0.5% and 10% for Mutation and 95% and 100% for Crossover, shown below against the Mean of the full data set at the end of each run.

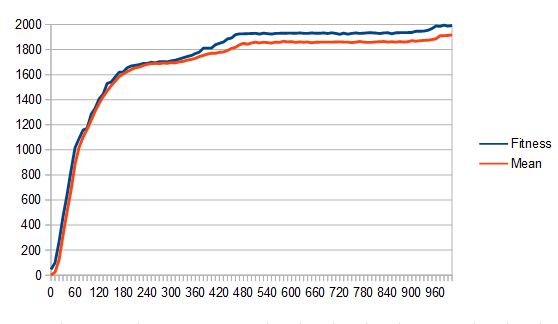
Mutation against mean fitness 

Crossover against mean fitness

Mutation has a clear winner at 1.5% Mutation rate. Crossover on the over hand doesn't seem to have a strong impact on the mean fitness, however there is a noticeable bump at 98% which can be used.



Running against 1.5% Mutation and 98% Crossover for 10 runs, the best individual came out at 1636 fitness with 7 rules. Due to the large length of the genome the resulting individual is put in the appendix as figure 1. The classification rate is only 81%, but the number of wildcards in the form of lower bounds of 0.0 and upper bounds of 1.0 are high, implying there is notable structure, therefore another set of 10 runs was done with a population of 2000 instead to try and get the strongest individual.



With the higher population count we achieved an almost perfect classifier with a 1992 fitness and 8 rules. Again, the individual is printed under the appendix, this time as Figure 2

**4 CONCLUSIONS**

In Data Set 1 there was a fairly low classification rate of just 75%, with a rule count of 8. Inside this rule set there is a notable lack of wildcards in the first 6 rules. This leads us to believe that there isn't much/any structure in the data given in Data Set 1.  
  
Data Set 2 however resulted in very reliable runs with 100% fitness and the minimum of 5 rules, implying that there is an easy to see structure to the data that was given.  
  
Data Set 3 produced an individual with a classification rate of 99.6%. Again there are many present wildcards in the form of lower bounds of 0.0 and upper bounds of 1.0, implying that it found structure in the data and did not over fit. This is further proven by the fact that training data was used unlike Data Set 1 and 2, which had too small a data set to do so for.  
  
The implementation of GA used was written in Python. Sadly, it wasn't until Data Set 3 that the much slower compute time for the foreach loops were made evident. This made the time to gather the data much slower than if it had been implemented in a lower level language such as C or a precompiled language such as Java.

**REFERENCES**

Freitas A.A. (2005) Evolutionary Algorithms for Data Mining. *Data Mining and Knowledge Discovery Handbook*.

Hong, T., Lee, Y. & Wu, M. (2014) An effective parallel approach for genetic-fuzzy data mining, *Expert Systems with Applications*, 41 (2), pp. 655-662.

Verma, G. and Verma, V. (2012) Role and applications of genetic algorithm in data mining. *International journal of computer applications*, 48 (17), pp.5-8.

Yassi, M. & Moattar, M.H. (2014) Robust and stable feature selection by integrating ranking methods and wrapper technique in genetic data classification, *Biochemical and biophysical research communications*, 446 (4), pp. 850.

Shah, S. & Kusiak, A. (2007) Cancer gene search with data-mining and genetic algorithms, *Computers in Biology and Medicine*, 37 (2), pp. 251-261.

## Appendix

### Figure 1 - Data Set 3 500 Pop

Best Individual - 1636 Fitness - 7 Rules

0.3301906, 1.0, 0.0, 0.98528814, 0.49898314, 1.0, 0.5006453, 1.0, 0.027047198, 1.0, 0.0, 0.90980184 - 1.0

0.32809347, 1.0, 0.59025866, 1.0, 0.08931512, 0.94302, 0.0, 0.9612043, 0.5208666, 0.8835254, 0.10159842, 1.0, 1.0

0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.51746356, 0.0, 0.864491, 0.5000885, 1.0 - 0.0

0.0, 0.52208966, 0.101033606, 0.9553545, 0.10293738, 0.9169944, 0.22674033, 0.78936267, 0.18460053, 0.7599559, 0.40441737, 1.0 - 1.0

0.0, 1.0, 0.0, 0.7592308, 0.0, 0.6435913, 0.030607153, 0.99425817, 0.0, 0.9937357, 0.0, 0.5787775 - 0.0

0.4821748, 0.7712494, 0.20122649, 0.4582235, 0.73198295, 0.88126624, 0.29800725, 0.8828433, 0.90065116, 0.4255965, 0.53821576, 0.49781933 - 1.0

0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0 - 1.0

### Figure 2 - Data Set 3 2000 Pop

Best individual - 1992 Fitness - 8 Rules

0.50144947, 1.0, 0.0, 1.0, 0.5039872, 1.0, 0.0, 0.99585664, 0.0, 1.0, 0.0, 0.5047414 - 1.0

0.49704492, 1.0, 0.0, 1.0, 0.0, 0.9947343, 0.0, 1.0, 0.0, 0.50366986, 0.012248697, 1.0 - 0.0

0.0, 0.5019123, 0.0, 1.0, 0.0, 1.0, 0.0, 0.4997641, 0.0, 1.0, 0.49886012, 1.0 - 0.0

0.09268573, 0.21469176, 0.41521934, 0.92445225, 1.0, 0.5199668, 0.36388066, 0.8037075, 0.2668232, 0.4170747, 0.97619534, 0.6679286 - 0.0

0.49395302, 1.0, 0.4131267, 1.0, 0.0, 0.62897706, 0.041396994, 1.0, 0.13174622, 1.0, 0.008339545, 0.49297637 - 0.0

0.6820637, 0.6468477, 0.15296312, 0.34163922, 0.062408783, 0.2414223, 0.29062086, 0.37652668, 0.54837304, 0.67251825, 0.36082414, 0.3434367 - 0.0

0.0, 1.0, 0.0, 0.50304204, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.490053 - 0.0

0.0, 1.0, 0.0, 1.0, 0.0025751055, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0 - 1.0

### Figure 3 - Project Link

https://github.com/colin969/Biocomputation