Thomas Borel 10018807 Bsc Computer Science

Researching the applications of genetic algorithms in data mining and classification

ABSTRACT

The paper aims to show how Genetic Algorithms can be efficiently used for data mining and classification problems. Through the use of an algorithm which is inspired by Darwinian evolution, rule bases can be optimized to classify large data sets with near perfect precision. The following describes the gradual process which has been adopted for achieving the latter. An experimentation section with algorithms of variable learning parameters demonstrates different attempts at classifying 3 separate data sets, which increase in complexity. A new algorithm is designed and implemented for each one, introducing more sophisticated evolution strategies as the data sets become more difficult to classify.

1 INTRODUCTION

The genetic algorithm can be categorised as an evolutionary algorithm, which is a subset of machine learning techniques inspired by evolution theory.

The algorithm is based on Herbert Spencer's concept of 'survival of the fittest'. The biological phenomenon of evolution means that genes which allow an individual to be more adapted to it's environment are much more likely to persist throughout multiple generations than genes that do not. Although genetic algorithms make use of the theory of evolution, the inspiration is only loose, and in reality the process is far more complex than the implementation of a genetic algorithm. The basic steps of the algorithm are carried out as follows:

An initial population is generated at random, the individuals of which are made of genes which represent potential solutions to the given problem.

An evaluation function determines an individual's fitness, which represents how apt the individual is in solving the problem at hand, either partially or fully.

Individuals are then selected to be reproduced. The higher the fitness of an individual, the more likely it is to be selected for the current generation of individuals to go through the reproduction process. Tournament selection is a popular and efficient selection process. A subset of the population is randomly selected, and the fittest individual of the subset is selected. By default, the tournament size should be approximately $1/10^{th}$ of the size of the entire population.

The selected population is then able to reproduce in order to form new individuals, through the use of two genetic operators. Crossover involves the recombination of two parents' genetic structure, forming two new individuals, and occurs for every member of the population. Uniform crossover is employed throughout this paper, a method by which each gene of an individual has a probability of being exchanged with it's partner's genes at the same position

Mutation is a much rarer phenomenon, during which a small portion of a single individual's genes may undergo a slight change

The population is then reevaluated, and the process is repeated until termination criteria

has been satisfied, such as an optimal solution has been reached, or a specified number of generations have passed.

Elitism is a useful mechanism which is essential to any efficient Genetic Algorithm. The concept is straightforward; at each iteration of the algorithm, the fittest individual of the population is recorded, prior to running the crossover and mutation operators. This allows the best fitness of the population to remain constant, and never in decline.

The algorithm described can achieve significant results in the field of data mining, and is easily implemented. In this case, an individual's genes are made up of potential rules for classifying a dataset, and the fitness function evaluates how many data entries an individual can successfully classify.

2 BACKGROUND RESEARCH

The idea of of applying evolutionary concepts to the field of computing can be traced back to 1948, however Holland named and popularised the concept of the Genetic Algorithm [1] in the 1970s

More specifically to the field of data mining, The Pittsburgh approach to evolving rule bases was theorised by Smith [2] and involves using single individuals to represent an entire potential rule base. This directly opposes the Michigan approach in which individuals represent single rules, and thus only make up partial solutions to the task at hand. The Pittsburgh method offers the advantage of easily avoiding rule repetition, as populations of Michigan type individuals have a tendency to converge towards similar rules as the algorithm progresses. The Pittsburgh model will therefore be adopted throughout the experimental section of this paper.

Genetic Algorithms have been widely used for data classification problems, as an alternative to the more traditional techniques such as clustering and nearest neighbour classifiers.

In the field of medicine, Ngan et al. [3] implemented a genetic algorithm in order to form complex Bayesian networks which classified large amounts of data surrounding limb fracture and scoliosis cases. In addition to the basic evolutionary operators, the system utilized the concept of cross validation, a well known statistical validation technique which involves separating a dataset for training and testing purposes. This notion will be employed extensively throughout this paper.
Similarly, Papadopoulos et al [4] successfully applied a Genetic Algorithm for classifying cancerous and healthy cells using data gathered from mammographies.

Research has shown that more complex evolution strategies can significantly improve the efficiency of evolutionary algorithms. Particularly self adaptive techniques which see the incorporation of probabilities such as mutation into the evolutionary process have proven to be beneficial[5]

Holland developed a more advanced strategy in using evolutionary search in data classification, with the Learning Classifier System [6] .The LCS combines reinforcement learning and genetic algorithms for the discovery and learning of condition: action rule based systems. The LCS operates in two phases:

The rule evaluation phase, during which a rule is selected based on an input string, and the rule's predicted payoff. Once an has been carried out, it's associated rule is rewarded based on the accuracy of the action.

The rule discovery phase: A genetic algorithm is run, with the predicted payoff of a rule being assigned as it's fitness. The genetic

operators are applied to the population, in the hope that new individuals will be formed with a higher predicted payoff.

3 EXPERIMENTATION

3.1 Data Set 1

3.1.1 Data Set and Algorithm Description

The first data set consists of 32 lines of data, each of which describe 6 digit long binary strings. The first 5 bits represent the input, whereas the final digit describes an action. This implies that a simple binary representation for the algorithm's individuals is sufficient for the first data set; genetic bits are stored in an integer array and can take the form of either a '0' or a '1'. In computational terms, the Java integer array data type is used, where the value of the array at a specific index represents a single gene

Simple data structures have been used to encapsulate the algorithms core functions; the Individual class contains data such as the individual's genes and fitness, and methods for applying the mutation operator and fitness function. Following the Pittsburgh approach, an individual's genetic material consists of multiple potential rules to be evaluated based on the data set and evolved. Individuals have a binary encoding, meaning that their genes can have one of two values, '0'1 or '1'. The Population class extends an ArrayList of Individual objects and contains functions which return the fittest individual of a population, and the population's total fitness. Separate classes have been written for achieving the selection process functionality, and the crossover genetic operator.

The code's algorithm executes as follows:

An initial Population object is constructed, which in turn constructs Individual objects containing randomly generated genes. The fitness function is applied to each individual in turn: A java for loop reads single lines of data at a time, and compares the input digits to the corresponding input digits of each potential rule contained in an individual's genes. If these do not match, the same comparison is applied to the next portion of the individual's genes. In the case in which the compared input strings match, the single action bits are in turn compared: equal action bits award a fitness incrementation to the individual. At this point, the loop continues to the next data string as opposed to the next portion of an individual's genes; This is essential as it penalises invalid rules, and allows the algorithm to avoid single rule repetition within the same individual. The whole process is repeated for each line in the data set, and in turn for each individual in the current population.

Once the evaluation process has been completed, the fittest individual of the population is determined and stored in a separate variable, to ensure elitism throughout the generations

Following this, the tournament selection method is called, with the Population object passed in as a parameters. The method returns the selected individual to the algorithm.

A Crossover object is constructed and the Population object is passed into it's main method, returning the same population of individuals with recombined genetic material. The method implements the uniform crossover algorithm: a global crossover rate is assigned to the population, and for each gene index of a pair of individuals, a random value is generated. If the latter is inferior or equal to the crossover rate, the individuals exchange their genes at the specific index.

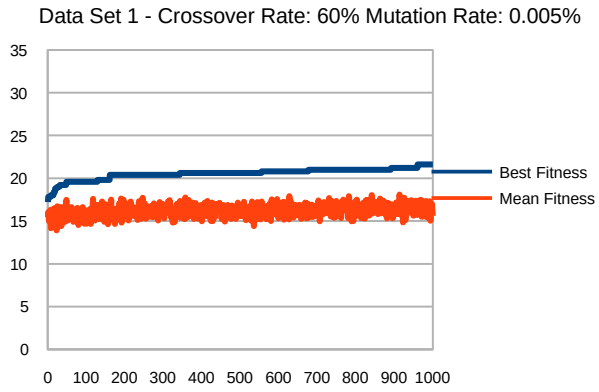
Similarly to this the mutation operator is applied to each individual. Each bit in an individual's genes has a chance of mutating based on a mutation rate. A '0' gene mutates to a '1', and vice versa.

The process described is repeated until

the algorithm has repeated 1000 iterations. The maximum fitness an individual can achieve is equal to the number of data entries being classified, 32. If an individual's fitness reaches this value, the algorithm has successfully classified the entire data set

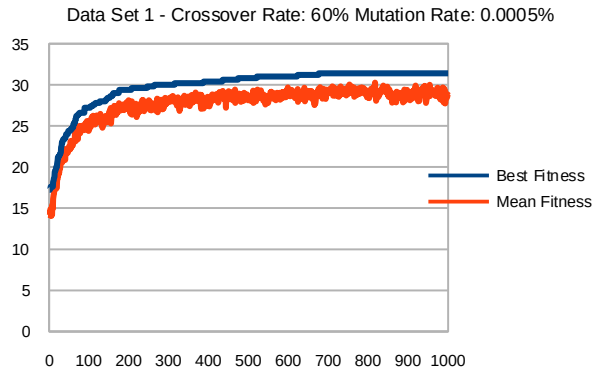
It is important to save some useful statistics surrounding the current population at a given generation, such as the population's mean and best fitness. This allows the algorithm's progress to be tracked, meaning the user can experiment with different parameters such as the mutation and crossover rates, and the population size. The modification of these variables can have subtle yet significant effect on the evolution process. In most cases a specific combination of these variables produce the best results, and need to be determined by experimentation.

3.1.2 Results Analysis

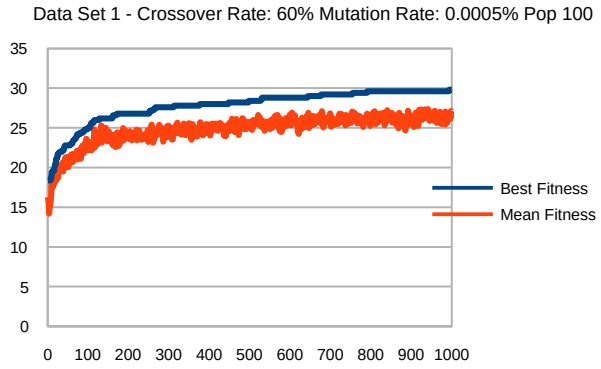


An initial experiment with a population size of 50, and crossover and mutation rates of 60% and 0.005% respectively yields uninteresting results; The best fitness of the population does not come near to reaching its best possible value, and the mean fitness remains stagnant. The shape of this graph implies that the population has converged prematurely towards a local optimum. This would mean that the population's genetic structure has stagnated in an area of the search space which contains a solution of average quality. As the latter is likely to be a local optimum, it's surrounding neighbours would be of inferior quality, meaning that the population is unlikely to move towards a different area of the search space.

The deduction made here is that the mutation rate is too high – as the average fitness in this case is stagnant, too many sudden changes in the genetic structure of the individuals mean that higher quality solutions are unlikely to be found. By applying mutation in a more incremental fashion, a much more progressive convergence of the global population can be achieved:



With the configuration shown in figure 2, the optimal solution, with a fitness of 32, has been achieved 3 out of 5 times.



The final experiment on this data set produces less significant results: Population convergence is less sudden, and the best fitness end result is inferior to the previous experiment. This is likely due to epistasis, a phenomenon by which the portion of an individual's genes which contributes to high fitness is recombined during the crossover process, resulting in a lowered fitness in the produced children. As the population here is more diverse than the previous experiment, portions of rules which classify large amounts of the dataset are likely to be broken up and not appear in future generations. This is further suggested by the mean fitness of the previous graph, which remains stagnant very early on in the experiment.

The configuration used in the second experiment of this section should therefore be considered optimal.

3.2 Data Set 2

3.2.1 Data Set changes

The data set is now made up of 64 lines, each containing strings of 7 digits formed of 6 input bits and a single action bit. Due to the larger volume and variation of the data set, generalisations can be made surrounding the behaviour it describes. Rules can be formed which allow for classification of the data

3.2.2 Program Modifications

In order to modify the program to accommodate the concept of generalisation some of the basic functions of the algorithm have been modified:

Gene generation: This must now introduce generalisations into the genetic structure with an equal probability as the other two values (0 and 1). in order to keep the integer array data structure which was used in the previous version of the program, it is represented as a '2'. It is essential that this modification does not affect the final bit of every rule in an individual's genes, as generalisations can only be made for the input of a rule, and not it's action.

The fitness function: When comparing a data entry to an individual's genes, selecting a '2' at any given index of the genes must now match both a '0' and a '1' at the same index in the data entry.

The mutation function must now have an equal probability of mutating a single bit in the integer array to a '2'. Again this must not affect a rule's final bit.

In addition to optimizing the quality of the rules in question, it is also beneficial to evolve the amount of rules an individual contains, in order to achieve a final rule base which describes the data set in as few rules as possible. Further program modifications must be made to achieve this:

Upon the initial random generation of an individual's genes, a randomly assigned integer determines the rule length of an individual, between minimum and maximum values defined and modifiable by the user.

Throughout the selection process, in a case where two individuals are equally the fittest

of the tournament, the shortest one is returned.

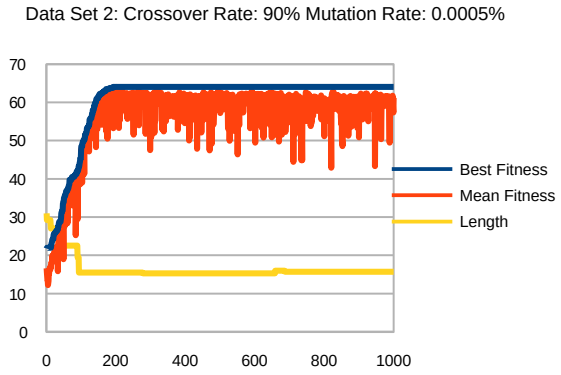
The crossover and mutation functions must be modified in order to accommodate individuals of variable length

After mutation, an individual also has a probability of incrementing or decrementing its length, provided this change is within the defined range. The probability of this phenomenon is also a configurable algorithm parameter.

Selecting the fittest individual of a population must also take length into account; In a similar fashion to the modified selection function, if several individuals are found to have the highest fitness of the population, the shortest is always returned.

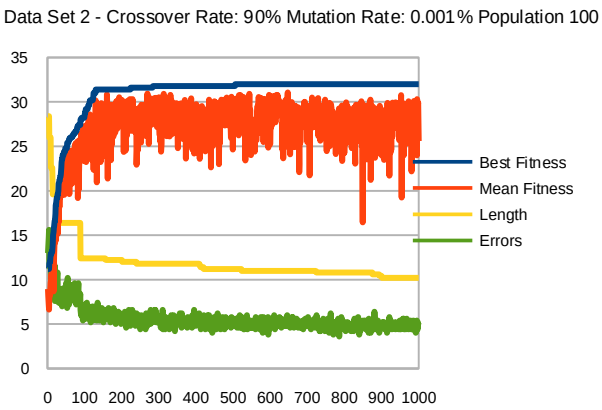
3.2.3 Experiments and results

Using a low mutation rate, the algorithm converges towards the maximum fitness within 200 generations. The crossover rate does not affect the evolution process significantly, provided that it is between 60% and 90%, and that it keeps the global population diverse. The maximum achievable fitness here is 64



At this point, the notion of cross validation can be introduced, which randomly halves the data set into separate training and testing data. This allows large datasets to be classified in shorter amounts of time. Providing a significant amount of generalisations can be made, a genetic algorithm should be able to classify a data set in such a way with minimal error

As individual length is an important factor, a less rapid convergence can allow a the population to examine areas of the search space which classify the dataset in fewer rules. In the case of the following graph, on average the best individual can classify the data set in 10 rules as opposed to 13 in the previous experiment. However, this is slightly at the detriment of solution convergence, as genetic mutation is less incremental



With an optimal parameter configuration, it is possible to classify the data set with 6 rules, with an average of 4 generalisations per rule, and 4 errors recorded:

220122 : 0
122022 : 0
202221 : 1
020222 : 1
121222 : 1
122202 : 1

3.3 Data Set 3 (and UCI data)

3.3.1 Data Set changes

The third data set no longer features binary data entries, but is now made up of 6 floating point values between 0 and 1 which make up the input string, and a single binary action bit.

3.3.2 Code Modifications

Now that individuals can longer have binary representations, the algorithm must be modified to make use of the float array data structure to represent genes. The modifications include:

Modifying the genetic structure of the individuals. Genes are now made up of two java float arrays, at each index a random value between 0 and 1 is initialized, with the exception of indexes which are multiples of 7 which are still randomly assigned a 0 or a 1 (the action bit). These arrays represent boundaries for the rules, meaning that the randomly initialised genes are seeded in such a fashion that one array is made of lower boundaries, and the other made of upper boundaries.

Individual evaluation. Upon comparing a potential rule contained within an individual's genes to a data entry, both the upper and lower boundaries at a given index are selected. If the data entry's value at the same index is contained within the two boundaries, the data entry has been matched by the individual, and the algorithm proceeds to the next floating point value. As previously, providing all of the data entry input values are contained within the rule's boundaries, it's final bit is evaluated. A fitness incrementation is awarded if the latter is equal to the rule's action bit.

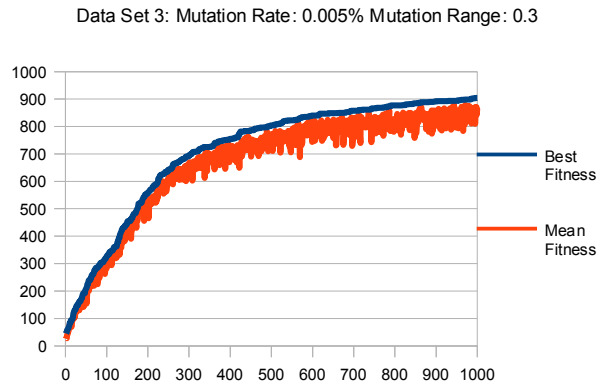
The mutation operator. This is changed to a much subtler mechanism: If a gene mutation occurs, a random floating point value is generated between 0 and a new configurable parameter, the mutation range. The generated value is randomly added or subtracted to the upper or lower boundaries, provided the mutated results remain within the range of 0 and 1. This means that rules which make generalisations at specific indexes will have lower and upper boundaries which stretch towards 0 and 1 respectively, allowing them to match many data values at the given index.

The rest of the core algorithm functions remain unmodified, albeit syntax modifications to adjust to the new float array data structure. Similarly to the previous algorithm, it is also beneficial to allow the algorithm to evolve the number of rules an individual contains.

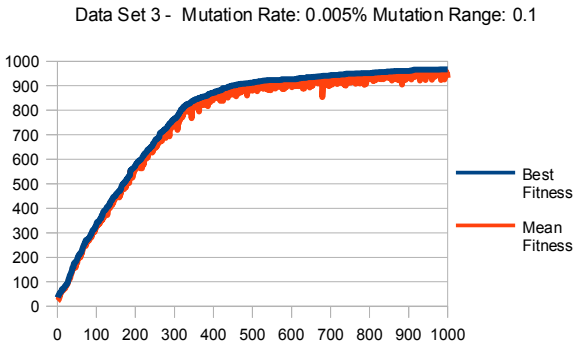
3.3.3 Experimentation

As the affect of applying various mutation rates has already been demonstrated, the focus in this section will be put on the mutation range parameter, a floating point value which determines the maximum value that a gene can be mutated by.

Again, the data is halved into training and testing data, making the maximum achievable fitness 1000. By applying the relatively high mutation range of 0.3, the following results are achieved



A dramatic fitness increase in the first 100 generations is observed, however, this begins to stagnate by the 250th generation. By the end of the experiment, the results are unsatisfying, as the algorithm has converged prematurely, and struggled to successfully classify 90% of the data.
By lowering the mutation range value, a much steadier evolution pace is observed



With a mutation range of 0.1, the algorithm shows no signs of stagnation until the 400th generation, and on average converges towards a best fitness of 974, the highest recorded best fitness being 983.

This is primarily due to the algorithm's ability to apply large mutations to the individuals during an exploration phase. As the algorithm progresses, current individuals with a high fitness need to be fine tuned to maximise their potential of covering more data entries, in a phase known as exploitation. In order for individual exploitation to take place, much smaller mutations need to be applied to it. These will haver a much higher probability of occurring if the mutation range is smaller

The progression of the number of rules and test errors are not represented here, due to the large scale of the graphs. It has been noted that the shortest individual length recorded is 8, with approximately 5 errors when cross validating the data. Similarly to the previous data set, the length of the fittest recorded individual is large during the early stages of the process, but declines rapidly.

4 CONCLUSIONS AND FUTURE WORKS

This paper shows that the Genetic Algorithm can produce precise rule bases when attempting to classify large datasets, making it highly competitive with the more traditionally used data mining algorithms

Although the Genetic Algorithm is based on stochastic elements, the evolutionary process never achieves it's target randomly. As shown in this paper, it is necessary to carefully select learning parameters in order to drive the process in a sustained and controlled fashion. It has been observed that while crossover allows the population to remain diverse, rule discovery usually happens during mutation. Generally speaking, mutation should be a rare occasion, as applying the operator too heavily can lead to epistasis. Furthermore, the population size should reflect the complexity of the problem to solve, and the individuals which make up the

potential solutions. Population size should increase to account for a more varied data set and ensure diversity.

In future work,more advanced evolutionary strategies should be implemented. Particularly, a self adaptive mutation operator could be employed. This would involve encoding the mutation rate into each individual's genotype This allows the algorithm to evolve the mutation rate throughout an experiment, as opposed to applying a static rate throughout the entire process. When generating the initial population randomly, each individual is assigned a random value between 0 and 1 as it's mutation rate. A new probability is defined for the algorithm, and is used as the rate at which to apply mutation to each individual's mutation rate. This occurs after crossover, before the genetic mutation operation, the functionality of which remains the same. This allows the algorithm to modify it's behaviour throughout it's course, applying large mutations when possible, and smaller mutations when the fine tuning of already fit individuals is required

A Baldwinian effect could also be implemented: This would allow the algorithm to achieve greater results when large areas of the search space need to be sampled. The Baldwin effect is based on the theory that individuals can adapt to their environment throughout their lifetime. This does result in the adapted genes being passed on to future generations, but the individuals who are more likely to adapt are also likely to persist through multiple generations. In computational terms this translates to individuals having a probability of running a local search algorithm, prior to their evaluation, based on a rate known as plasticity. This is another stochastically assigned parameter which can also be mutated

REFERENCES

[1] Holland, J.H. (1973) Genetic Algorithms and the optimal allocation of trials.*Siam Journal of Computing*.

[2] Smith, S.F. (1980) A learning system based on genetic adaptive algorithms.*PhD Thesis, University of Pittsburgh*.

[3] Ngan, P.S., Wong, M.L., Lam, W., Leung, K.S. and Cheng, J.C.Y. (1999) Medical data mining using evolutionary computation. *Artificial Intelligence in Medicine 16*., pp. 73-96.

[4] Kontos, K. and Maragoudakis, M. (2013) Breast cancer detection in mammogram medical images with data mining techniques. *IFIP AICT*., pp. 336-347.

[5] Back, T. (2000) Evolutionary computation 2: advanced algorithms and operators. .

[6] Holland, J.H. (1976) Adaption. *Progress in Theoretical Biology*.