Evolutionary Algorithm for Classification Data Mining

Maxim Nethercott 15005627

GITLAB: https://gitlab.uwe.ac.uk/m2-nethercott/Genetic-Algorithm-Problem-Solving.git

1 INTRODUCTION

With the huge amount of data being generated in the world every day, at a rate far higher than it can be analysed by human comprehension alone, data mining becomes an extremely important activity for extracting as much useful information from this data as possible.

In this paper I describe the process of evolving a system that correctly classifies a given set of input variables. I used three data sets. Data set one contains 32 data items, consisting of five binary variables producing a binary output. Whereas Data set 2 expanded on this, containing 64 data items, consisting of 6 binary variables producing a binary output. Finally, data set 3 contained 2000 data items, consisting of 6 floating point numbers as variables with a binary output. This meant that the matching method used would need to be altered to accommodate these differences. The idea was to evolve my own set of rules to effectively match the data set, thus the emphasis on classification. Ideally as few rules as possible would be used, so that a pattern in the data could be spotted.

Throughout the weeks, I implemented different techniques to try and fine tune the Data Mining algorithm in reaching its goal, of classifying the input data. This included, adding wildcards so that certain rules in the population of individuals could match with more than one rule in the input data. Thus achieving a more efficient and powerful classification algorithm as described by Eiben and Schoenauer (2002). Furthermore, implementing different forms of selection, crossover and mutation whilst varying parameters would result in interestingly varied results.

Genetic algorithms (Holland, 1975, 1992) provide a method to perform randomized global search in a solution space. The common underlying idea behind all variations of evolutionary algorithms is that, given a population of individuals, the environment causes natural selection and this produces a rise in the fitness of the population. Given a quality function to be maximized, we can randomly generate a set of candidate solutions and apply the quality function as a fitness measure as reported by Marque, Garci and Sa'Nchez (2013) in their wider literature review. Based on this, some of the better candidates are chosen to seed the next generation by applying selection, mutation, and crossover operators. These operators lead to a set of new candidates that compete with the old ones for a place in the next generation. This process can be iterated until a candidate with sufficient quality is found or a previously fixed computational limit is reached.

2 BACKGROUND RESEARCH

Evolutionary Computing has been used for Classification Data Mining in many ways. As the field of Artificial Intelligence and Intelligent Systems grows further, more and more innovative advances will be made to propel the subject further.

One such use for Evolutionary Computing is in Pre-processing of Streaming Data as described by Desale and Ade (2015). In real life scenarios, pre-processing is a very important factor of the data mining process, because real data comes from a very complex environment and is often incomplete and redundant. Desale's and Ade's interesting research describes a new approach for using a genetic algorithm as an evolutionary search method with feature selection. This proposed system is applied on two stream datasets and their results are analysed. From their experimental results it can be concluded,

that the proposed method helped in selecting the minimum number of features from both datasets. This improved the accuracy of the system when classifying between normal and anomalous behaviour.

It's worthy of note that although the system in this research combines evolutionary computing with other methods, the basic premise behind the genetic algorithm is the same, highlighting the fact that the Genetic Algorithm is a very flexible system to use.

Evolutionary computing has also been used to assist in credit scoring. The goal of an exercise by Finlay (2006) was to apply a Genetic Algorithm to create a linear scoring function. Although other types of scoring functions exist such as a multi-layer perception, a linear function was considered appropriate because linear scoring rules have continuing popularity with credit scoring practitioners.

In Finlay's experiment, both binary and integer encoding schema were considered, which is interesting as binary encoding was present in the work relating to this report. The results show that the final parameter values are as follows: population sizes for which 'good' solutions were found was greater than 400, Crossover rate was 0.5, mutation rate was 0.003. It's clear that a good selection of the GA parameters improves both computation time and solution accuracy. Work by Roeva, Fidanova and Paprzycki (2013) show that the use of smaller populations results in lower accuracy of the solution despite being obtained for a smaller computation time. However, an increase of the population size increases the accuracy of the solution. This effect is observed up to a certain point, when the use of a larger population size does not improve solution accuracy and only increases the needed computational resources. They conclude that this is something that must be carefully thought about in all Genetic Algorithms to ensure success.

In this model, the results show that the system did have the propensity to outperform other approaches in some situations. However, in other situations the performance was worse than that of the competitor models. The results would suggest that while a GA approach can produce models that are competitive with traditional methods that include linear regression, logistic regression and neural networks, they do not significantly outperform these methods, and in some cases perform less well. Finlay (2006) concluded that it may be the case that alternative formulations of the GA could lead to better solutions, and this view is supported by the comparisons of the models he derived from using binary and integer encodings.

A rather interesting and potentially lifesaving use of evolutionary computing is described by Tan, Heng and Lee (2003). In this paper, a two-phase hybrid evolutionary classification technique is proposed to extract classification rules that can be used in clinical practice for better understanding and prevention of unwanted medical events.

Interestingly, in Tan, Heng and Lee's classification task the discovered knowledge is represented in the form of IF-THEN classification rules. This is very comparable to the task that for which this report is written. Such rules state that the presence of one or more items implies or predicts the presence of other items. IF-THEN rules have the advantage of being a high level and symbolic knowledge representation that contributes to the comprehensibility of the discovered knowledge. In Tan, Heng and Lee's case the IF<given set of rules> THEN <class of medical problem>.

Tang, Heng and Lee (2006) also concluded that a careful examination of the relationship between the predictive accuracy of a rule set and its number of rules reveals an interesting finding. The rule sets with a large number of rules will not necessarily lead to high predictive accuracy, although they generally provide good performances on the training sets. It can also be observed that the first few rules in a rule set often cover a large portion of the samples and leave relatively few samples for the remaining rules. Therefore, when the dataset is not noise-free, a large number of rules may cause over-fitting and leads to poor generalization.

The results from the evolutionary classifier in Tan, Heng and Lee's research have been validated against hepatitis and two breast cancer datasets. Simulation results showed that the proposed system produces comprehensible and good classification rules for the three medical data sets.

3 EXPERIMENTATION

3.1 Data Set 1

Data set one contained 32 rules, using binary representation, each rule containing five variables and one output. Due to the seemingly random pattern within this Data set, I was unable to effectively evolve a rule set that could produce a fitness of 32, even when introducing wildcards.

Example of rules matching with data set, using wildcards. (described later)

Evolved rules:	Data Set:
1, 1, 2, 0, 0 0	1, 1, 1, 0, 0 0
1, 1, 1, 1, 1 0	1, 1, 1, 1, 1 0
1, 0, 0, 1, 0 1	1, 0, 0, 1, 0 1
0, 1, 0, 2, 1 0	0, 1, 0, 1, 1 0
2, 2, 0, 0, 0 0	0, 0, 0, 0, 0 0
1, 0, 1, 0, 1 0	1, 0, 1, 0, 1 0

Firstly, I created a population of genes with integers ranging from 0-1, thus allowing for binary representation. This would then be evaluated by the fitness function.

Fitness-based selection is the force that represents the drive toward quality improvements in an EA as described by Eiben and Schoenauer (2002). They state that first important feature about fitness computation is that it represents 99% of the total computational cost of evolution in most real-world problems. Secondly, the fitness function very often is the only information about the problem in the algorithm. They assert that any available and usable knowledge about the problem domain should be used. Therefore, the fitness function is vital in moving the GA towards the correct classification and thus must be thought about carefully.

For Dataset 1 the fitness function was relatively simple, in that, if a chromosome in the individuals rule base matched with one of the variables from the data set and the output was the same then fitness for that individual would be incremented. This would happen on a linear scale, iterating through both data set and rule base of an individual.

Succeeding this we would then enter a while loop, which would continue evolving the individuals in the population, using certain genetic operators, until some form of stop condition was met.

Firstly, Tournament Selection occurs. Tournament Selection is increasingly being used as a GA selection scheme because it is simple to code and is efficient for both nonparallel and parallel architectures, as shown by Miller and Goldberg (1995). Furthermore, Tournament Selection can also adjust the selection pressure, to adapt to different domains, by simply increasing or decreasing the tournament size. Miller and Goldberg show how Tournament selection provides selection pressure by holding a tournament among all the individuals in the population. The winner of the tournament is usually the individual with the highest fitness. The winner is then inserted into the offspring. The offspring, being comprised of tournament winners, has a higher average fitness than the average population fitness. Miller and Goldberg conclude that this fitness difference provides the selection pressure, which drives the GA to improve the fitness of each succeeding generation. The offspring is then put back into the population and its fitness is then evaluated.

Single Point crossover is then used to further allow variation in the population. This works by selecting a common crossover point in the population and then swapping the corresponding genes. For example, in my GA two parents are selected for crossover and then a random split point is assigned and combining the parents at crossover point creates two offspring. The premature convergence in a GA can be avoided by selecting appropriate breeding operators, such as crossover. The crossover rate controls the capability of GAs in exploiting a located hill to reach the local optima. The higher the crossover rate, the quicker

exploitation proceeds. Lin, Lee and Hong (2003) show that a crossover rate that is too large would disrupt individuals faster than they could be exploited. Crossover is explorative; it makes a big jump to an area somewhere "in between" two (parent) areas. Lin, Lee and Hong say that typical values for crossover are set between 0.5 and 1.0. Therefore, the crossover rate was set at 0.8.

Again, the fitness function is then used to calculate the fitness for this new population.

The mutation operator is used to change some elements in selected individuals, using a mutation rate of X, leading to additional genetic diversity to help the search process escape from local optimal traps. Mutation is exploitative, it creates random small diversions, thereby staying near the parent.

NOT logic was used, as this works well with the binary representation. Therefore, if the mutation rate satisfied the IF condition, if the gene was a 0 it would be made a 1 and vice versa. This is basically Bitwise bit-flipping with a fixed probability This would be made slightly more complicated with the introduction of wildcards; however, this will be explained in Data Set 2. Again, the fitness function is then used to calculate the fitness for this new population.

The process described above occurs until the GA has met its goal state or until a set number of generations is achieved.

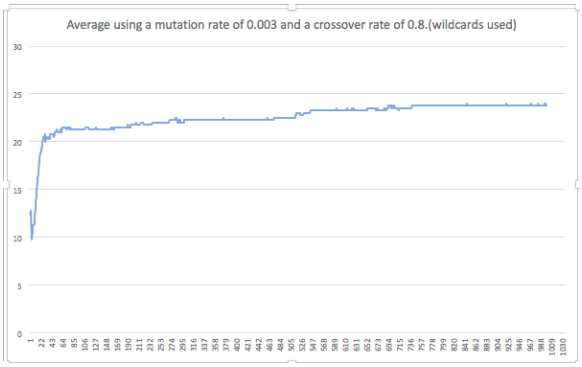

Figure 1: Results as average behaviour over more than one run. Mutation Rate 0.003. Population size:600.

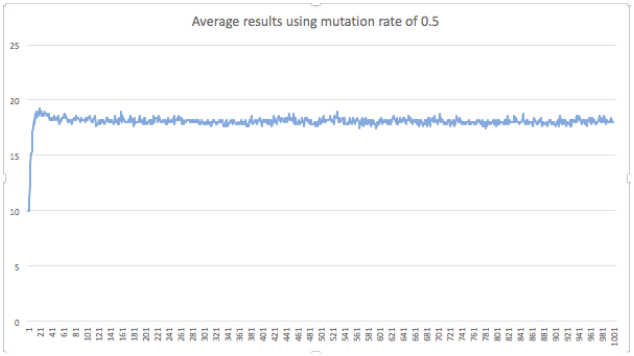

Figure 2: Results as average behaviour over more than one run. Mutation rate 0.05. population size 600.

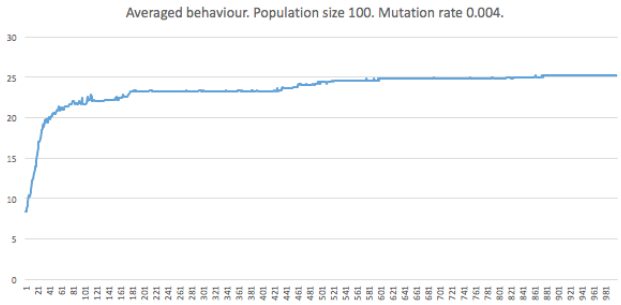

Figure 3. Results as average behaviour over more than one run. Mutation rate 0.004. population size 100.

The graphs demonstrate aspects of what has been previously discussed above. Figure one shows that with a large population, of 600, and a 'low' mutation rate, a reasonably good fitness can be achieved. However, with the same population size and a high mutation rate the GA seems to get stuck in local optima. This could be down to too much variation. A high population size is deemed beneficial, to a point, as it can increase the accuracy of the GA, however a smaller mutation rate is required as less variation is needed due to the increased population already bringing this variation. What more, the line in figure 2 is very jumpy, highlighting the fact that the high mutation rate is leading it too far from the parent. As the mutation rate allows the GA to explore the landscape in a random manner.

In contrast, figure 3 shows that with a lower population size and a slightly increased mutation rate, the GA can evolve the rules slightly quicker and on average can achieve a slightly higher fitness.

3.2 Data Set 2

Data Set 2 contained 64 data items. Again using binary representation, each item consisted of 6 binary variables producing a binary output. The Genetic Algorithm was able to successfully classify all 64 rules in this data set, often very efficiently with a rule base of 10. Importantly, by gradually decreasing the number of rules in the rule base, a pattern was revealed in the data.

In figure 4 a rule base of 5 was used and using wildcards for generalisation, was successfully evolved to match all 64 items in the data set. The first 2 genes in the variable is used as an index to produce the output. For example: 0,0 in binary is 0 therefore the value at location 0 will be the output. A further example: 1,1 in binary is 3 therefore the value at position 3 is 0 and so the output will be 0.

0, 0, 2, 2, 2, 0, =0
1, 1, 0, 2, 2, 2, =0
0, 1, 2, 2, 0, 2, =0
1, 0, 2, 0, 2, 2, =0
2, 2, 2, 2, 2, 2, =1

figure 4: showing the pattern in dataset 2.

Adding wildcards is based on Schema Theorem. John Holland introduced the notation of schema (Holland, 1975). A schema is a set of bit strings of one, zero and 2's in this case. Where the 2 can assume either value 1 or 0. The ones and zeros represents the fixed positions of a schema, whilst 2's represent wild cards. Genetic Algorithms manipulate schemata when they run. A chromosome matches a schema when the fixed positions in the schema match the corresponding positions in the chromosome. Due to the fitness function making the probability of reproduction proportional to chromosome fitness, a schema with above average fitness will tend to occur more frequently in the next generation of chromosomes, and a schema with below average fitness will tend to occur less frequently.

Adding Wild cards meant altering the mutation operator. A switch statement was used to implement the logic. Using the same IF statement as before, there were now three cases that could occur: If the gene was a 0, then it had an even chance of being mutated to a 2 or a 1, if the gene was 1, it had an even chance of being mutated to a 2 or a 0 and finally if the gene was 2 then it had an even chance of being mutated to a 1 or a 0. However, output must stay the same and even though there was an IF statement in place to stop the output being mutated to a 2, due to crossover, a 2 could still occur in the output. Therefore, a further switch statement was implemented that made sure the output was only mutated between 0 and 1.

Finally, the matching function inside of fitness evaluation had to be tweaked to allow for the wild card. Instead of looking to match data set with rules directly, it was adjusted to allow for generalisation. For example: if rule equals data item or rule is equal to 2 then increment match count.

One of the most commonly used chromosome selection techniques is the roulette wheel selection as explained by Zhong, Hu, Gu and Zhang (2005). My work decided to implement this technique to see how the results would differ from tournament selection. Each individual is given a slice of a circular roulette wheel. The area of the slice within the wheel is equal to the individual's fitness ratio. Therefore, Individual's with a higher fitness will be more likely to be selected. It is like spinning a wheel where each individual has a segment on the wheel proportional to its fitness. The roulette wheel is spun, and when the arrow comes to rest on one of the segments, the corresponding chromosome is selected.

The way this was implemented is as follows: A FOR loop was implemented where the total fitness was calculated. A random number between 0 and the total fitness was generated and whilst looping through the population this number was reduced by each individual's fitness until it was less than or equal to 0. The corresponding individual was then selected.

Figure 5 (below) shows the convergence using tournament selection. This relatively smooth line represents the genetic algorithm reaching maximum fitness around 130 generations. In contrast, figure 6 (below) showing roulette wheel selection highlights the fact that this method of selection struggles to get much better fitness values than 50, even after the full

200 generations. This indicates that the Genetic Algorithm converges more efficiently using tournament selection and achieves a smoother result in the process.

Work by Razali and Geraghty (2011) suggests that Tournament Selection gives a chance to all individuals to be selected and thus preserves diversity, moreover it causes a low susceptibility to takeover by dominant individuals. On the other hand, although in theory roulette wheel selection does preserve diversity, outstanding individuals will introduce a bias in the beginning of the search that may cause premature convergence and loss of diversity. They assert that this will prevent the population from exploring other potentially better individuals. Furthermore, if individuals in a population have very similar fitness values, Razali and Geraghty say it will be difficult for the population to move towards a better one since selection probabilities for fit and unfit are very similar. To conclude, this may explain the results from figure 6. As the algorithm seems to have premature convergence at the beginning and then get stuck, as it isn't exploring other potentially better individuals.

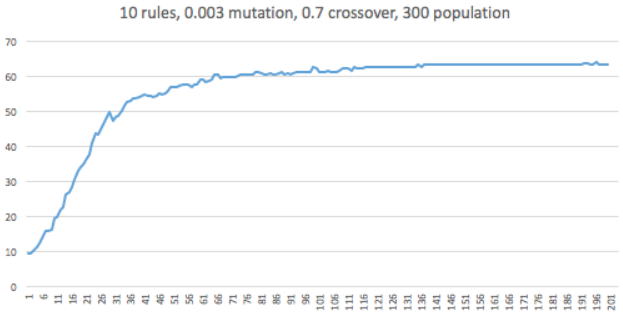

Figure 5: results as average behaviour over more than one run. Tournament selection.

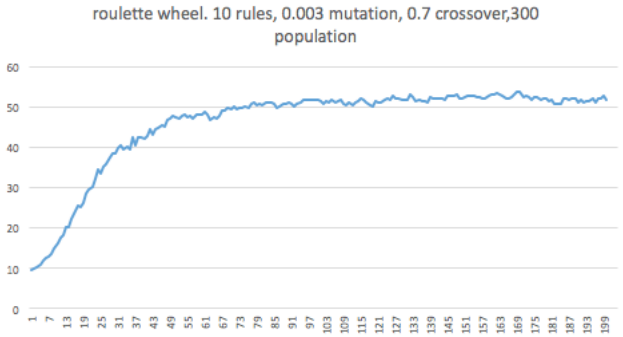

Figure 6: results as average behaviour over more than one run. Roulette wheel selection.

3.3 Data Set 3 (and UCI data)

Data set 3 contained floating-point chromosome representation. With this representation there is no need for an explicit encoding mechanism. Therefore, the genetic operators do not handle bit strings and are defined in a different manner. For example: the mutation operator does not randomly change one bit, but randomly chooses a floating point number within a particular range.

To deal with the change of representation it was decided that the generated rules would contain 12 floating-point variables and 1 binary output. Therefore, this would allow a boundary for the data item to sit within granting this to be the main premise of the fitness function. Thus the gene size was 2 times the variable size, plus the output size, times the rule size. ((2*6)+1) * rule size.

The matching function inside of fitness was changed to accommodate the change in representation. For each variable in the data, there was two rules acting as the boundary. Whilst looping through both rules and data, if data was in-between the corresponding rules this would result in incrementing the count. If the counter reached 6 then this would indicate there was a successful match corresponding to rule and data.

Mutation was also altered to deal with floating-point representation. Whereas before, mutation only had to flip certain bits between 0,1 and the wildcard, it is now made slightly more complicated. A random float was declared and if the gene selected plus this random float is less than one, this will become the new gene. However, if this number is bigger than one then if the gene selected minus this random number is bigger than 0 this will become the new gene. The advantage to this approach is that it offers better control, and often better performance than flipping bits in binary encoding.

When decreasing the number of rules to 5 and running the algorithm for many generations until a high fitness was found a pattern was exposed in the

rules. The 1st and 6th variables act as an index for the condition in the middle of the variables.

- RULE 1: If the first value is greater than 0.5 and the sixth value is smaller than 0.5 then the 3rd value must be greater than 0.5. With an output of 1.
- RULE 2: If the first value is greater than 0.5 and the 6th value is greater than 0.5 then the 5th value is greater than 0.5. With an output of 1.
- RULE 3: If the 1st value is smaller than 0.5 and the 6th value is smaller than 0.5 then the 2nd value is greater than 0.5. With an output of 1.
- RULE 4: If the first value is smaller than 0.5 and the 6th value is greater than 0.5 then the 4th value is greater than 0.5. With an output of 1.
- Else the output will be 0.

A comparison of genetic algorithm and multi-layer perceptron can be made to learn more about the efficiency and accuracy of this problem. Learning in a multilayer perceptron involves a training set of input patterns to be presented to the network. The network computes its output pattern and if there is an error, the weights are adjusted to reduce this error. Figure 8 shows that the multilayer perceptron is able to correctly classify 79.9% of the data that is very similar to the Genetic algorithm, shown in figure 7.

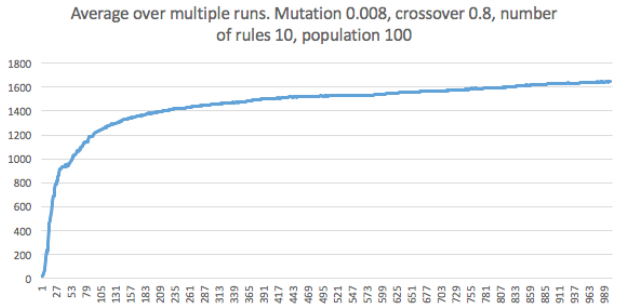

Figure 7: Results as average behaviour over more than one run.

```
=== Summary ===

Correctly Classified Instances        1598               79.9    %
Incorrectly Classified Instances       402               20.1    %
Kappa statistic                          0.5992
Mean absolute error                      0.2748
Root mean squared error                  0.3782
Relative absolute error                 55.0357 %
Root relative squared error             75.6906 %
Total Number of Instances             2000
```

Figure 8: Multilayer perceptron using training data. Weka

```
=== Summary ===

Correctly Classified Instances         307               76.75   %
Incorrectly Classified Instances        93               23.25   %
Kappa statistic                          0.5332
Mean absolute error                      0.272
Root mean squared error                  0.3821
Relative absolute error                 54.4483 %
Root relative squared error             76.4169 %
Total Number of Instances              400

=== Detailed Accuracy By Class ===
```

Figure 9: Multilayer perceptron. Using 80% split, classifying on untrained data. Weka

A multilayer perceptron can often be much more complicated to design and produce than a Genetic algorithm and so the similar results suggest that the Genetic Algorithm, for this task, is favourable due to its simplicity to design and produce.

Figure 9, shows the performance of the multi-layer perceptron on unseen data, quite often called validation data. The perceptron was trained with 80% of the input data, leaving it 400 items to classify. It managed to correctly classify 307 items. This is again similar to the performance of the Genetic Algorithm, thus proving the credibility of both methods.

4 CONCLUSIONS

Genetic algorithms are a heuristic solution search technique inspired by natural evolution. They are a robust and flexible approach that can be applied to a wide range of learning and optimisation problems. Firstly, the concept behind the algorithm is very easy to understand and it will nearly always produce an answer that will get better over time.

Genetic algorithms are intrinsically parallel. Most other algorithms are serial and can only explore the solution space to a problem in one direction at a time, and if the solution they discover turns out to be suboptimal, there is nothing to do but abandon all work previously completed and start over. However, since GA has multiple offspring, they can explore the solution space in multiple directions at once. If

one path turns out to be a dead end, they can easily eliminate it and continue work on more promising avenues, giving them a greater chance each run of finding the optimal solution.

Most importantly, tuning the parameters such as selection, crossover, mutation and population size play a detrimental role in the efficiency of the algorithm. It became clear that there are many different ways of implementing these operators and so careful selection and consideration over which method to implement will result in better performance.

The results revealed that tournament selection is more efficient in obtaining highest fitness with less number of generations compared to roulette wheel selection. However, this may be only applicable for a small problem size. When the size of the problem increases, in data set 3, tournament selection as well as roulette wheel selection becomes susceptible to premature convergence.

In future, to make my genetic algorithm more effective and efficient it can be incorporated with other techniques within its framework to produce a hybrid genetic algorithm that can reap best from its combination. Hybrid genetic algorithms have received significant interest in recent years and are being increasingly used to solve real-world problems quickly, reliably and accurately without the need for any forms of human intervention. (Bajpi and Manoj Kumar,2008)

REFERENCES

Bajpai, P. and Manoj Kumar, M. (2008) Genetic algorithm – an approach to solve global optimization problems. *Indian Journal of Computer Science and Engineering*. 1 (3), pp. 199-206.

Desale, K. and Ade, R. (2015) Preprocessing of Streaming Data using Genetic Algorithm. *International Journal of Computer Applications* [online]. 120 (17), pp. 16-19. [Accessed 11 November 2017].

Eiben, A.E. and Schoenauer, M (2002) Evolutionary Computing. *Information Processing Letters* [online]. 82 (1), pp. 1-6. [Accessed 08 November 2017].

Finlay SM (2006). Using genetic algorithms to develop scoring models for alternative measures of performance. *Technical Report 2006/022, Department of Management Science,* Lancaster University, UK.

Holland JH (1975). Adaptation in Natural and Artificial Systems. University of Michigan Press: Ann Arbor, MI.

Holland JH (1992). Adaptation in Natural and Artificial Systems. MIT Press: Cambridge, MA.

Lin, W.Y., Lee, W.Y. and Hong, T.P. (2003) Adapting Crossover and Mutation Rates in Genetic Algorithms. *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING*. 19 (1), pp. 889-903.

Marque ´ s, A.I., Garcı ´ a, V. and Sa ´ Nchez, J.S. (2013) A literature review on the application of evolutionary computing to credit scoring. *Journal of the Operational Research Society* [online]. 64, pp. 1384-1399. [Accessed 12 November 2017].

Miller, B.L. and Goldberg, D.E. (1995) Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*. 9, pp. 193-212.

Razali, N.M. and Geraghty, J. (2011) Genetic Algorithm Performance with Different Selection Strategies in Solving TSP. *Proceedings of the World Congress on Engineering*.

Roeva, O., Fidanova, S. and Paprzycki, M. (2013) Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. *Proceedings of*

the 2013 Federated Conference on Computer Science and Information Systems., pp. 371-376.

Tan, K.C., Yu, Q., Heng, C.M. and Lee, T.H. (2003) Evolutionary computing for knowledge discovery in medical diagnosis. *Artificial Intelligence in Medicine*. 27, pp. 129-154

Umbarkar, A.J. and Sheth, P.D. (2015) CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW. *ICTACT JOURNAL ON SOFT COMPUTING*. 6 (1), pp. 1083-1092

Zhong, J., Hu, X., Gu, M.I.N. and Zhang, J. (2005) Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms. *International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce*.