REPORT

Classifier Learning with Genetic Algorithms

Christabella Bastian (5002465), John Wilshire (3421072), Iwan Adhicandra (3199421)

COMP9417 Machine Learning and Data Mining, Assignment 2

1. Introduction

This report will explain our work on the project of classifier learning with genetic algorithms. In the field of artificial intelligence, Genetic Algorithm offers a method to knowledge that is based lightly on replicated growth. Hypotheses are frequently labeled by bit strings whose understanding be influenced by the application. The examination for a suitable hypotheses originates with a population, or group, of preliminary hypotheses. Participant of the existing population contribute the increase to the following generation population by means of procedures, for example, crossover and random mutation, which are designed after procedures in evolution. At each period, the hypotheses in the existing population are assessed comparative to a given amount of fitness, with the best suitable hypotheses chosen probabilistically as seeds for creating the next generation. However, working on dynamic and varied data sets is challenging, as genomes start to unite prematurely near solutions, which may not be usable for future data. Therefore, the goal of this project was to implement the basic Genetic Algorithm method including the point mutation rule and crossover rules, such as single-point, two point, and uniform. Two standard datasets from University of California Irvine including balance-scale and mushroom are used for testing. There are also graphical outputs showing how the system works including the selection process to construct a new generation of hypotheses. The report is organized as follows. Section 2 will describe the background theory for constructing genetic algorithm. Section 3 explains the method used for learning classification including the data sets and the designed genetic algorithm. The results are presented in Section 4. Then, some related works are presented in Section 5. Finally, Section 6 summarizes the paper.

2. The Overview of Genetic Algorithm

This section will describe the theoretical background for constructing genetic algorithm.

Crossover

This will create two new offspring from two parents. Parameters for operators are chosen randomly at each application. In the crossover operation, two solutions are shared to form two new solutions. The parents are selected from the population by a fitness function of the results. Some approaches occur for choosing the solutions for the crossover procedure. Normally, we can use a probability constructed from the fitness of the result or by rank. In rank selection, selection is established from the rank of the fitness values of the results of the population. The construction of the offsprings from the crossover process is accomplished by removing the crossover portion of the first parent and then injecting the crossover fragment of the second parent. Then, the second offspring is formed in a symmetric way.

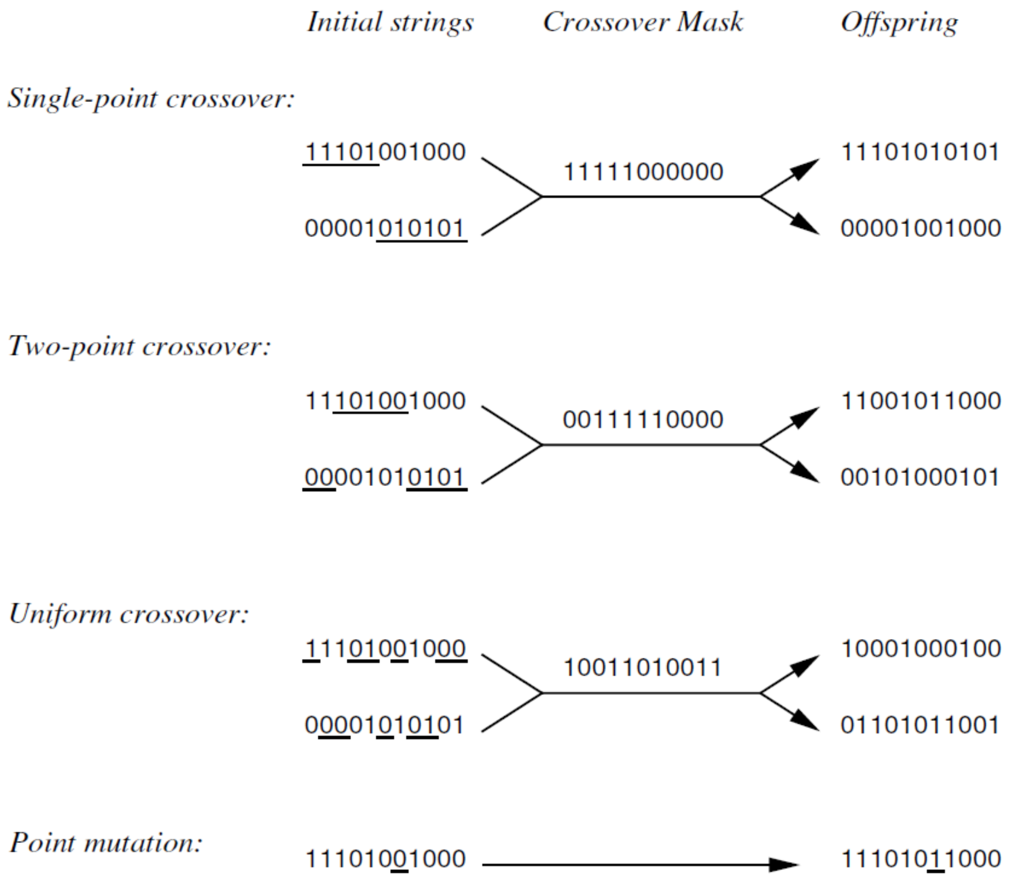
Mutation

This is a new type of single parent. There are two possible types of mutations. First, a function can only substitute a function. Second, a complete subtree can substitute another subtree.

Fitness Function

The fitness function describes how good a program can solve the problem.

The diagram below shows variations of common GA operators [5].



3. Method

This section will describe the methods used in implementing our basic genetic algorithm. It starts with the used datasets, and then the constructed algorithm.

3.1 Datasets

From the project's requirement, two open sample datasets, namely balance scale and mushroom, which were acquired from the University of California Irvine machine learning repository [6], are implemented to assess feature collection routine over an extensive series of data. Both datasets differ broadly along with the amount of data instances, the amount of features existing (both discrete and numeric), the amount of classes existing in the data, the variance in size among the largest and smallest classes in the data, and the nonexistence or existence of lost data values.

3.1.1 Balance Scale

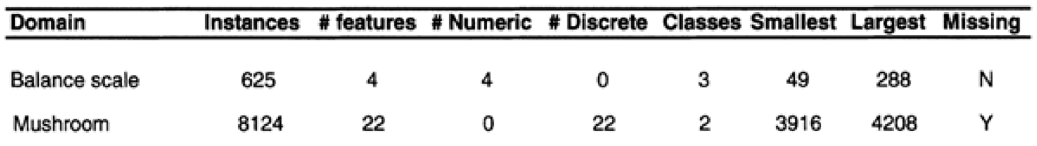
The features of this datasets [6] consist of *Right-Distance: 5 (1, 2, 3, 4, 5), Right-Weight: 5 (1, 2, 3, 4, 5), Left-Distance: 5 (1, 2, 3, 4, 5), Left-Weight: 5 (1, 2, 3, 4, 5), and Class Name: 3 (L, B, R).* The balance scale dataset has an artificial noise-free problem with 625 samples. Each sample is categorized as requiring the balance scale tip to the left (288 samples, 46.08 %), be balanced (49 samples, 7.84 %), or tip to the right (288 samples, 46.08 %). The features are the right distance (RD), the left distance (LD), the right weight (RW), and the left weight (LW). The accurate method to discover the class is the bigger of (RD RW) and (LD LW). If they are identical, it can be said to be balanced. Since the logic for 3 potential results (right left, balance) is to be discovered, the main property for optimizing is the accuracy.

3.1.2 Mushroom

This data set [6] contains reports of hypothetical models matching to 23 types of gilled mushrooms. Every types is recognized as absolutely eatable, absolutely toxic, or of unidentified edibility and not suggested. This latter class was joined with the toxic one. There is no simple regulation for defining the mushroom's edibility. The features of this datasets consist of stalk-root, stalk-surface-below-ring, stalk-surface-above-ring, stalk-shape, stalk-color-below-ring, stalk-color-above-ring, cap-color, cap-surface, cap-shape, gill-attachment, gill-color, gill-size, gill-spacing, bruises, odor, veil-color, veil-type, ring-type, ring-number, spore-print-color, habitat and population.

The features of both datasets are summarized in Table 1.

Table 1: Datasets used for experiments



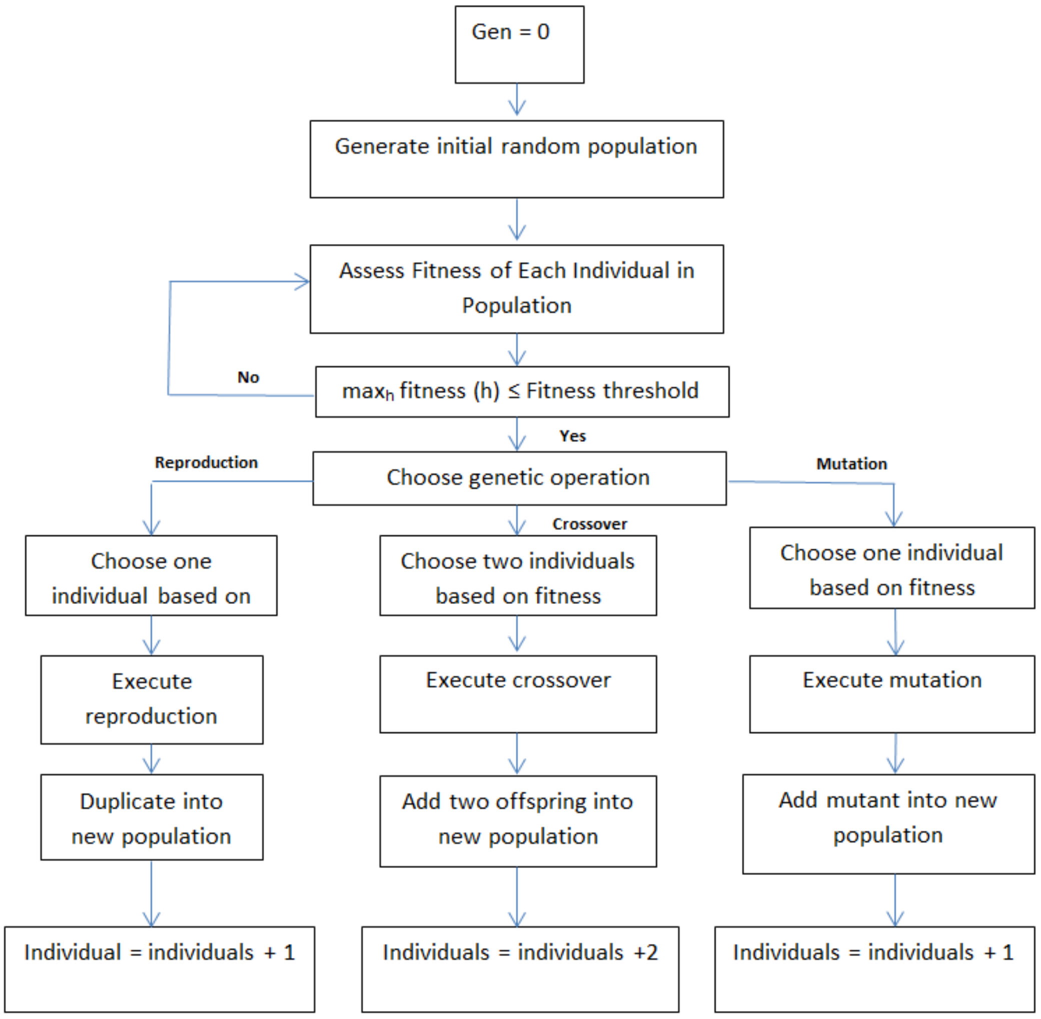
Domain	Instances	# features	# Numeric	# Discrete	Classes	Smallest	Largest	Missing
Balance scale	625	4	4	0	3	49	288	N
Mushroom	8124	22	0	22	2	3916	4208	Y

3.2 The Designed Algorithm

From the project's requirement, the designed algorithm must capture the key constructions of a Genetic Algorithm (GA) method for learning classification procedures. However, some constraints are required. These need to be provided by the user.

3.2.1 Flow Chart for our designed Genetic Algorithm

The flowchart and pseudocode below illustrate our designed algorithm for genetic algorithm. We modified the idea from [7] and [5] and tailored it according to our own criteria and objective.


Gen = 0
Generate initial random population
Assess Fitness of Each Individual in Population
No
max_h fitness (h) ≤ Fitness threshold
Yes
Reproduction
Choose genetic operation
Mutation
Crossover
Choose one individual based on
Choose two individuals based on fitness
Choose one individual based on fitness
Execute reproduction
Execute crossover
Execute mutation
Duplicate into new population
Add two offspring into new population
Add mutant into new population
Individual = individuals + 1
Individuals = individuals +2
Individuals = individuals + 1

3.2.2 Pseudocode for our designed Genetic Algorithm

```
GACV(p, r, m, fitnessfunc, fitnessTH, folds)
    ● Initialize: $P \leftarrow p$ random hypotheses
    ● Assess: for each $h$ in $P$, calculate $Fitness(h)$
    ● While $[max_h\ Fitness(h)] < Fitness\_threshold$ Do
        1. Select: Rank Selection by the number of selected population = (1-r)p
            • Sort all hypotheses by fitness
            • Probability of selection is comparative to rank
        2. Crossover: Probabilistically choose $\frac{r.p}{2}$ pairs of hypotheses from $P$. For each pair,
           $<h1, h2>$, create two offspring by applying the Crossover operator. Augment all
           offspring to $Ps$.
        3. Mutate: Invert a randomly selected bit in m · p random members of $P_s$
        4. Update: $P \leftarrow P_s$
        5. Assess: for each h in P, calculate $Fitness(h)$
Return hypothesis from $P$ with maximum fitness.
```

3.2.3 Data Set Preprocessing

In our design algorithm, we created a new set of instances, which is based on the old data set for balance scale (Task 1) instead of using [lw, ld, rw, rd, class]. So, we consider the following equation:

$$rw*rd - lw*ld = torque \qquad (1)$$

Therefore, the new data set becomes {torque, class}

$$Torque = rw.rd - lw.ld \qquad (2)$$

with: *rw = right weight; rd = right distance; lw = left weight; ld = left distance*

3.2.4 Hypotheses Encoding Structure

We designed the structure for encoding the hypotheses for both datasets (balance scale and mushroom) as the following tables. Each hypothesis is the result of concatenating set of hypothesis, (disjunctive set of rules), where each hypothesis is represented as bit-string encoded for each class. This method will guarantee that after learning process, rule for all class will be generated.

1. *balance-scale*

torque bitstring	left class bits	torque bitstring	balance class bits	torque bitstring	right class bits
e.g. 01……..101	*10*	*e.g. 01…….101*	*11*	*e.g. 01…….101*	*01*

2. *mushroom*

attribute values bitstring	edible class bits	attribute values bitstring	poisonous class bits
e.g. 01………………101	*0*	*e.g. 01……………..101*	*1*

3.2.5 Selection Method Illustration

In our designed algorithm, we illustrate selection method in the following description.

1. Single point crossover for balance-scale data

Parent 1: 10100100011000100011111000010110100001000001000001011001010011010010010000101101011010011100010000011100001011101100000111001001010101011111001001100100 1
Parent 2: 1110110101001000100011010001000010010001010100010101010110000000111101011100011000110111010000000000111010110111000101111000111101001101101101010001000 01

Offspring 1: 101001000110001000111110000101101000010000010000010110010100110100100100001011010110100 1110001000000111000010111011000001100011110100110110110101000100001
Offspring 2: 111011010100100010001101000100001001000101010001010101011000000011110101110001100011011 101000000000011101011011100010111110010010101010111110010011001001

2. Two-points crossover for balance-scale data

Parent 1: 010110000001000011100111111111111001010101010111011010110110010011100110010100011101001101 001010111010111000001011010101101011100111011101000101111011010001
Parent 2: 011001000111101001000111101011110010011000111101010000010000110100010111011000101011111 001110001001011111101100011111100101100000101101110110110101101101
Offspring 1: 011001000111101001000111101011110010011000111011010110110010011100110010100011101001101 001010111010111000001011010101100101100000101101110110110101101101
Offspring 2: 010110000001000011100111111111111001010101010111101010000010000110100010111011000101011111 001110001001011111101100011111101011100111011101000101111011010001

3. Uniform crossover for balance-scale data

Parent 1: 0110100010110111010011101111110010100111000001100110111000011101110111011110001011000001 001011100000111101000010100011000100100110001111111100001001111001
Parent 2: 0100011100001101000101010000000100010001100111010110100000101011000011100011100011011010 101110011100011100111001111110110100001101100001110111101100110101
Offspring 1: 01001101001001010001011110011000010010100001110011011000001111110111010111000011001010 001010101000111100001001101010000100101101000101110111001000110001
Offspring 2: 011000101001111101001100011000110010011100101010110101000101001000011101010101011010001 101111010100011101110010110111110100000110101011111100101101111101

3.2.6 Evaluation

There are two evaluation criterias used in this work including accuracy and validation.

3.2.6.1 Accuracy

To measure accuracy performance criteria, we assume that the standard definition used in binary classification. *TP* and *TN* denote the number of true positives and true negatives. *FP* and *FN* denote the number of disordered negative and positive cases. In term of two-class cases, the accuracy rate on the positives, known as sensitivity, is stated as *TP/(TP+FN)*, while the accuracy rate on the negative class, also known as specificity, is *TN/(TN+FP)*. Classification accuracy is simply [10]:

$$(TP+TN)\,/\,N \tag{3}$$

where, $N = TP+TN+FP+FP$ is the total number of cases.

3.2.6.2 Validation

To validate measurement, *k-fold* cross validation is usually used as a method for assessing the performance of an algorithm or a classifier. Assume that we have a set of *m* training examples. So, a single run of k-fold cross validation will continue as follows: Firstly, we need to assemble the training examples in a random order. Secondly, the training examples can be split into *k* folds. For example, *k* portions of about *m/k* examples individually. Thirdly, assuming that for $i = 1, \ldots, k,$ we need to do some steps, such as training the classifier using all the examples that do not belong to Fold *i,* testing the classifier on all the examples in Fold *i*, then, calculating n_i, the number of examples in Fold i that were incorrectly classified. Finally, we can return the subsequent estimation to the classifier error in the following equation [8]:

$$E = \frac{\sum_{i=1}^{k} n_i}{m} \tag{4}$$

To achieve an exact estimation to the precision of a classifier, k-fold cross validation is run numerous times, each with an altered random procedure in Step 1. Let $E_1, \ldots, E_t$ be the precision estimations gained in *t* runs. So, we can define the following equations [8]:

$$e = \frac{\sum_{j=1}^{t} E_j}{t} \tag{5}$$

$$V = \frac{\sum_{j=1}^{t} (E_j - e)^2}{t-1} \tag{6}$$

$$\sigma = \sqrt{V} \tag{7}$$

Therefore, the calculation for the algorithm performance is an error of *e* with standard-deviation of *σ.*

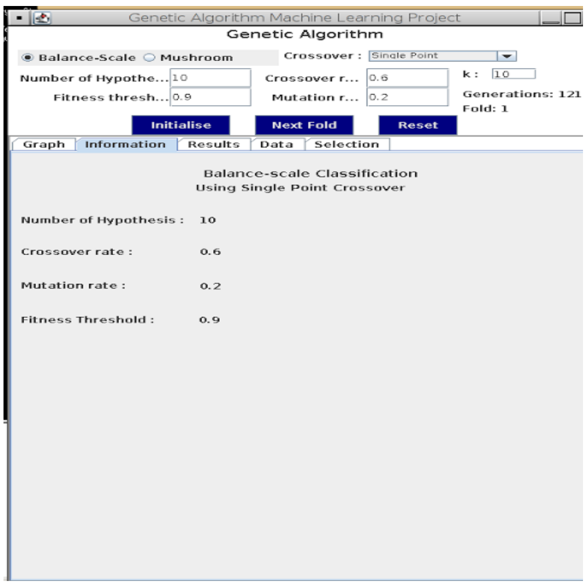
In all our experiment, we set initially some default parameters to run our design algorithm program such as the number of hypotheses, fitness threshold, crossover rate and mutation rate as shown in Table 2.

Table 2: Initial Parameters

Parameter	Value
The Number of Hypotheses	10
Fitness threshold	0.9
Crossover rate	0.6
Mutation rate	0.2

4. Results

This section will describe the results of using our genetic algorithm application for learning classification. In our design application, we created a GUI application showing how the system is learning. In addition, it also shows the key features of the algorithm such as the selection process to create a new generation of hypotheses. Initially, we inserted some default parameters, such as the number of hypotheses, fitness threshold, crossover rate, mutation rate and *k* as shown in Table 2. Figure 1 and Figure 2 show our GUI application for displaying information and population.

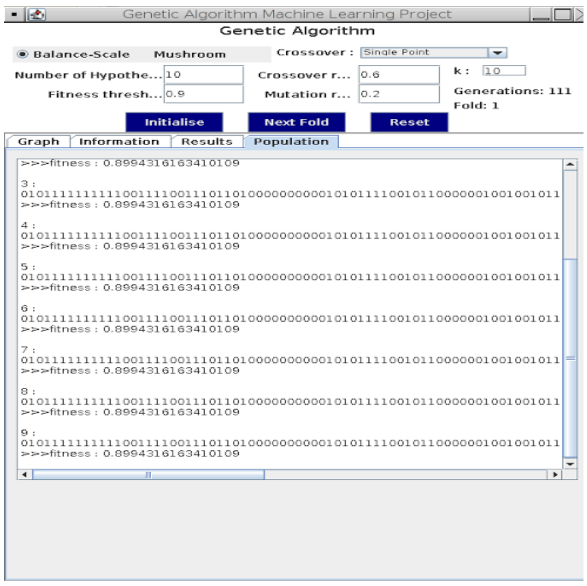


Genetic Algorithm

[✓ Balance-Scale] Mushroom
Crossover : Single Point
Number of Hypothe... 10
Fitness thresh... 0.9
Crossover r... 0.6
Mutation r... 0.2
k : 10
Generations: 121
Fold: 1

Balance-scale Classification
Using Single Point Crossover

Number of Hypothesis :	10
Crossover rate :	0.6
Mutation rate :	0.2
Fitness Threshold :	0.9



Genetic Algorithm

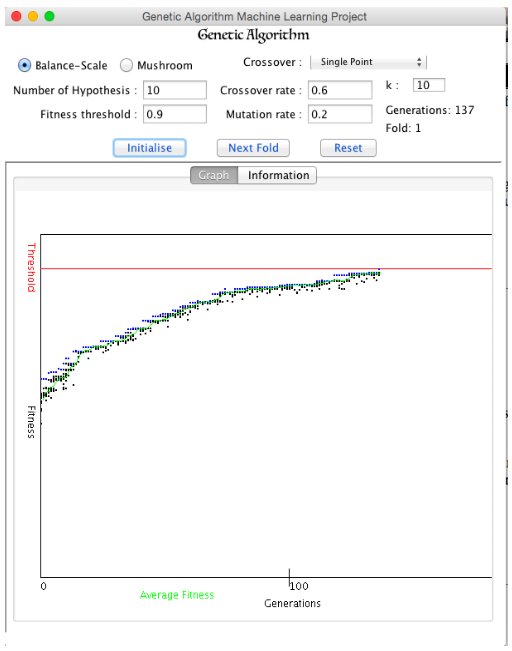
[✓ Balance-Scale] Mushroom
Crossover : Single Point
Number of Hypothe... 10
Fitness thresh... 0.9
Crossover r... 0.6
Mutation r... 0.2
k : 10
Generations: 111
Fold: 1

```
>>>fitness : 0.8994316163410109

3 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109

4 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109

5 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109

6 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109

7 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109

8 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109

9 :
0101111111111001111001110110100000000000101011110010110000001001001011
>>>fitness : 0.8994316163410109
```

Figure 1. GUI Application for Information

Figure 2. GUI Application for Population

We can also select the graph to show the trend. Figure 3 and Figure 4 show respectively the comparison of performance between using balance scale dataset with $k = 1$ and $k = 10$. We can see from the graph that the effects of increasing fold on balances scale do not have much impact from the initial setting.


Genetic Algorithm Machine Learning Project

Genetic Algorithm

[✓ Balance-Scale] Mushroom

Crossover : Single Point

Number of Hypothesis : 10

Crossover rate : 0.6

k : 10

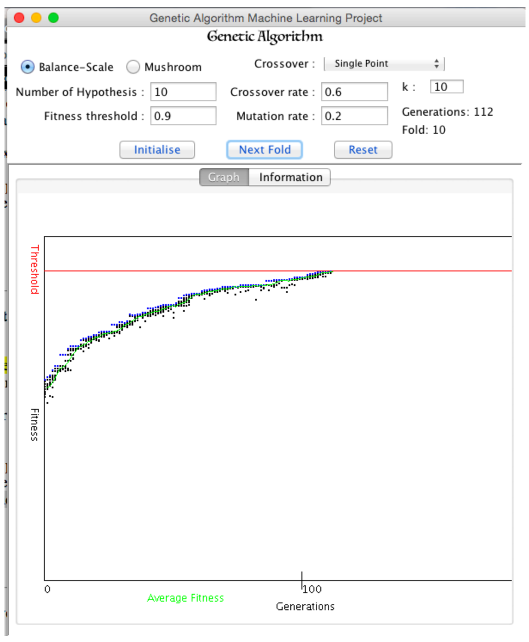
Fitness threshold : 0.9

Mutation rate : 0.2

Generations: 137

Fold: 1

Figure 3. Balance scale with fold = 1


Genetic Algorithm Machine Learning Project

Genetic Algorithm

[✓ Balance-Scale] Mushroom

Crossover : Single Point

Number of Hypothesis : 10

Crossover rate : 0.6

k : 10

Fitness threshold : 0.9

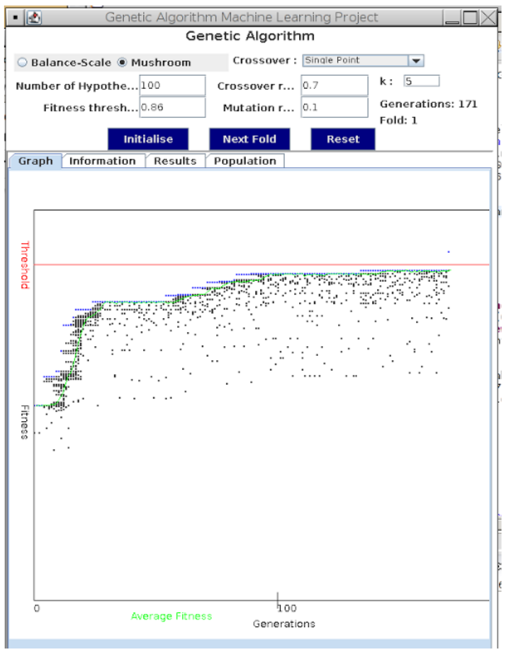
Mutation rate : 0.2

Generations: 112

Fold: 10

Figure 4. Balance scale with fold = 10

Figure 5 shows the application using mushroom dataset with p = 100. We can see from the graph that the population is scattered widely before reaching the threshold. It is predicted since the fact that the mushroom dataset have a varied and large dataset compared to balance scale dataset.


Genetic Algorithm Machine Learning Project

Genetic Algorithm

Balance-Scale [✓ Mushroom]

Crossover : Single Point

Number of Hypothe... 100

Crossover r... 0.7

k : 5

Fitness thresh... 0.86

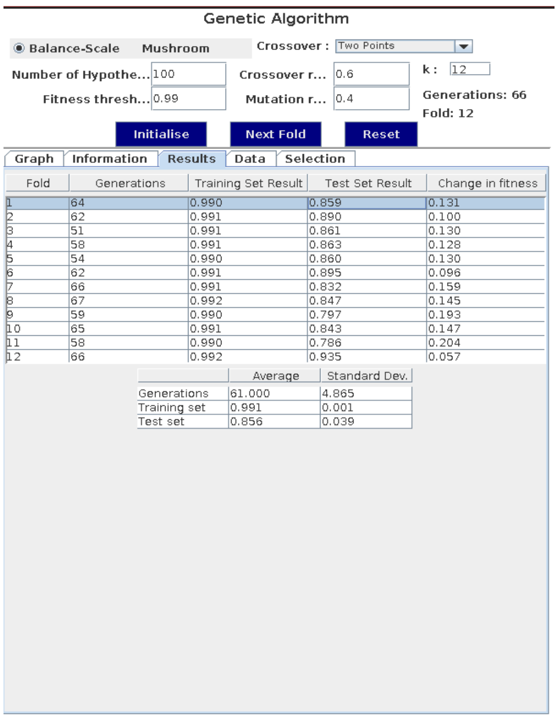
Mutation r... 0.1

Generations: 171

Fold: 1

Figure 5. Mushroom dataset with p = 100

Then, we estimated the classification accuracy of an algorithm using 12-fold cross validation. In our experiments, we reset the parameters with the following setting: the number of hypotheses is increased to 50, fitness threshold is increased to 0.99, mutation rate is increased to 0.4, *k* is increased to 12, and the selected crossover type is changed to two-points . The results shown in Figure 6 indicated that if using *p* = 100 the average is 61.00 and the standard deviation is 4.865. In contrast with if using lower population size, such as *p* = 50, the average increases to 81.167 and the standard deviation also increases to 7.116. So, by increasing the number of population the accuracy may reduce significantly.

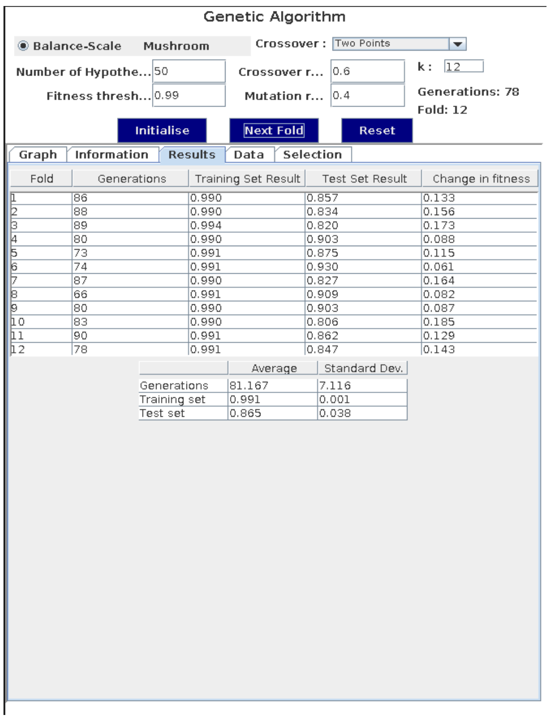


● Balance-Scale   Mushroom
Crossover : Two Points
Number of Hypothe... 100
Crossover r... 0.6
k : 12
Fitness thresh... 0.99
Mutation r... 0.4
Generations: 66
Fold: 12

Fold	Generations	Training Set Result	Test Set Result	Change in fitness
1	64	0.990	0.859	0.131
2	62	0.991	0.890	0.100
3	51	0.991	0.861	0.130
4	58	0.991	0.863	0.128
5	54	0.990	0.860	0.130
6	62	0.991	0.895	0.096
7	66	0.991	0.832	0.159
8	67	0.992	0.847	0.145
9	59	0.990	0.797	0.193
10	65	0.991	0.843	0.147
11	58	0.990	0.786	0.204
12	66	0.992	0.935	0.057

	Average	Standard Dev.
Generations	61.000	4.865
Training set	0.991	0.001
Test set	0.856	0.039

Figure 6. Balance Scale dataset with p=100, crossover type: two-points



● Balance-Scale   Mushroom
Crossover : Two Points
Number of Hypothe... 50
Crossover r... 0.6
k : 12
Fitness thresh... 0.99
Mutation r... 0.4
Generations: 78
Fold: 12

Fold	Generations	Training Set Result	Test Set Result	Change in fitness
1	86	0.990	0.857	0.133
2	88	0.990	0.834	0.156
3	89	0.994	0.820	0.173
4	80	0.990	0.903	0.088
5	73	0.991	0.875	0.115
6	74	0.991	0.930	0.061
7	87	0.990	0.827	0.164
8	66	0.991	0.909	0.082
9	80	0.990	0.903	0.087
10	83	0.990	0.806	0.185
11	90	0.991	0.862	0.129
12	78	0.991	0.847	0.143

	Average	Standard Dev.
Generations	81.167	7.116
Training set	0.991	0.001
Test set	0.865	0.038

Figure 7. Balance Scale dataset with p=50, crossover type: two-points

The graphs also show that if the algorithm is in steady condition for a particular dataset, the variance of the cross validation estimations should be about the same, independent of the number of folds specified. While the algorithms are not steady condition, they are about steady. Measuring *k*-fold cross validation with high *k* values will diminish the variance while growing the bias. As *k* declines and the sample sizes get reduced, there is variance due to the uncertainty of the training sets themselves, leading to a rise in variance. This is most obvious for datasets with numerous classes, such as mushroom. In these situations, stratification may help, but reiterated runs may be a better method. The results show that stratification is normally an improved structure including variance and bias, when associated to fixed cross validation. We can say that it is better to use stratified 12 fold cross validation for model selection. Therefore, the results have indicated that the problem using genetic algorithm with varied data sets may be improved by changing fitness function, increasing the amount of mutation, or by using selection methods that retain a different population of solutions.

5. Related Work

There have been few studies focusing on performance evaluation of using genetic algorithm with varying data sets. Firstly, Milidiu et al. [1] proposed an evolutionary approach based on Genetic Algorithms to automatically implement the template generation process. Then, Wang et al. [2] proposed a novel generative model for classification. They also presented an algorithm for learning and empirically evaluate it on large number of data sets. In addition, Kohavi [3] studied cross□ validation and bootstrap for estimating the accuracy and selecting the model. Finally, Peterson et al. [4] studied performance of genetic algorithm by comparing the selection performance using various fitness functions to achieve various accuracy and balance. Some of their work are related with our works since we also used the same datasets from UCI. However they focused mainly on implementing and comparing some of their own different classifier or algorithms.

6. Summary

This project has implemented a Genetic Algorithm, which is a method to learning that is based lightly on replicated growth. The goal of this project was to implement the basic Genetic Algorithm method including the point mutation rule and crossover rules such as single-point, two point, and uniform. In contrast to other classifier learning methods, the search for a successful hypothesis can be characterised as bit string encodings of classification rules. It is best viewed as a stochastic approximation to optimizing the fitness function, which is based on minimizing error. The results have indicated that the problem using genetic algorithm with varied data sets may be improved by changing fitness function, increasing the amount of mutation, or by using selection methods that retain a different population of solutions. In the future, further testing is required to improve knowledge in term of the general connection between uncertainties related to the cross-validation method. In addition, the outcome of the strength of algorithm on the dependability of cross-validation approximations should also be considered in further detail.

References

[1] R. L. Milidiú, J.C. Duarte, C. N. dos Santos, "*Evolutionary TBL template generation*", J. Braz. Comp. Soc. vol.13 no.4 Campinas Dec. 2007, http://dx.doi.org/10.1007/BF03194255

[2] Y. Wang, N.L. Zhang, T. Chen, and L.K.M. Poon, "*Latent Tree Classifier*", in proc. of Symbolic and Quantitative Approaches to Reasoning with Uncertainty: 11th European Conference, ECSQARU 2011, Belfast, UK, June 29-July 1, 2011

[3] Kohavi, R., "*A Study of Cross□Validation and Bootstrap for Accuracy Estimation and Model Selection*", International Joint Conference on Arti□cial Intelligence (IJCAI), 1995

[4] M. Peterson, M. Raymer, G. Lamont, "*Balanced Accuracy for Feature Subset Selection with Genetic Algorithm*s", Proc. of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), 3, 2514-2521

[5] Mitchell, T. "*Machine Learning*". McGraw Hill, 1997.

[6] Lichman, M. (2013). "*Mushroom data set*" and "*Balance Scale data se*t", UCI Machine Learning Repository. Irvine, CA https://archive.ics.uci.edu/ml/datasets/Mushroom , https://archive.ics.uci.edu/ml/datasets/Balance+Scale

[7] Genetic Programming Tutorial, http://www.geneticprogramming.com/Tutorial/

[8] Wu. J., "*k-fold Cross Validation*", Lecture Notes on Machine Learning, http://www.csie.ntu.edu.tw/~b92109/course/MachineLearning.html

[9] J. M. Barreiro, F. Martin-Sanchez, V. Maojo, F. Sanz, "*Biological and Medical Data Analysis*", Proc. of 5th International Symposium, ISBMDA 2004, Barcelona, Spain, Nov. 18-19, 2004

[10] Cohen, G., Hilario, M., and A. Geissbuhler, "*Model Selection for Support Vector Classifiers via Genetic Algorithms. An Application to Medical Decision Support*", ISBMDA 2004, LNCS 3337, pp. 200–211, 2004.

Appendice

README

Classifier Learning with Genetic Algorithms

Goal: implement basic genetic algorithm algorithm to learn classification rules.

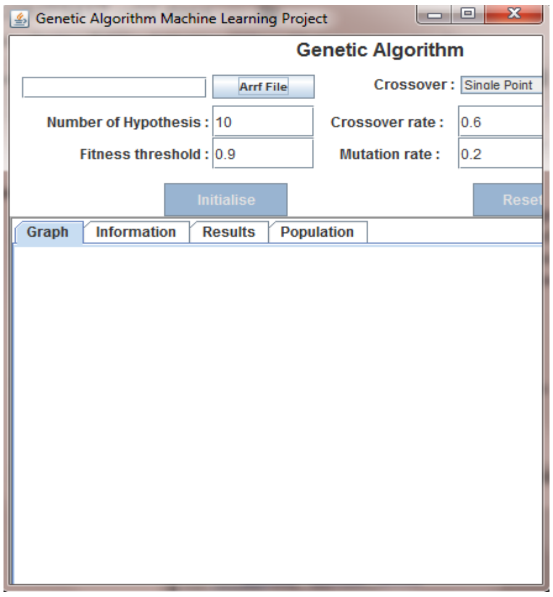
Packages

- **GeneticAlgorithm:** contains all files related to the algorithm (*Accuracy.java, Crossover.java, FitnessFunction.java, Hypothesis.java,* and *Validation.java)*.
- **gui:** contains all files related to the graphical representation (*GADrawer.java* and *GAGUI.java, GAResult.java*)
- **main:** contains test file to run the program (*GA.java* and *GACV.java*)

How to start

1. The submitted program requires the following system requirements:
   a. Windows XP or above, and Mac IOS
   b. Minimum Java SE 6
   c. Eclipse IDE for Java developer

2. To explore the full potentials of Genetic Algorithm, such as viewing the Genetic Algorithm progress in the various graphic forms, you will need a computer with high-end graphic card. To run the code, run *GA.jar*, and select the appropriate arff files, such as *balance-scale.arff* and *mushroom.arff* as the following graph.


Genetic Algorithm Machine Learning Project

Genetic Algorithm

Crossover : Single Point

Number of Hypothesis : 10

Crossover rate : 0.6

Fitness threshold : 0.9

Mutation rate : 0.2

Figure 1. GUI Application using *GA.jar*

or alternatively, we can extract *GA.jar* and use the following steps:

```
$ javac $(find GA.java)
```

To run the graph, compile:

```
$ javac $(find GAGUI.java)
```

3. Various fields in the control panel will be filled with default values from the database. You may choose to change four of the fields, for example, the number of hypotheses, crossover rate, mutation rate, and fitness threshold. Validation check will be performed to ensure that valid number are entered, for example, more than zero for the number of hypothesis, 0 (unlimited) and above for Number of Evolution, 0.00 to 1.00 for Fitness threshold, 0.00 to 1.00 for Crossover Rate, and 0.00 to 0.10 for Mutation Rate.

4. You may choose to manually change the crossover rate and mutation rate.

5. During a run, you can also view the progress in different forms by choosing one of the four tabs including Graph, Information, Result and Population.

6. If you click on the Result tab, the monitor will show the result of evolution progress by generation.

7. If you click on the Graph tab, a progress graph showing the fitness value versus generation will be drawn and shown on the monitor. The red line represents fitness threshold, the green line represents average while the blue one represents best fitness.