Experiment 2: Genetic Algorithms

Krish Sukhani
2018140059
Batch D

Aim:
To apply genetic algorithm for a given problem

Problem Statement:

Task 1: Find out a research paper based on Genetic Algorithm. The research paper should be on some practical application of a genetic algorithm in any one of the following domains:
1. Financial Domain
2. Computational Creativity (Music/Drawing/Painting)
3. Medical Domain
4. Security Applications
5. Video/Image Processing

Make a summary report of this paper with the following outline:
1. Problem that the paper solves
2. Description of the component involved in the GA:
    a. Type of encoding scheme used in the chromosome representation
    b. Type of selection, mutation and crossover schemes used in the paper
3. Results observed by the authors.
4. Observations and Conclusion

This report summary will be presented by the student in the first session.

Task 2: Implement the chosen research paper by using the same components mentioned in the paper. Also, use other types of components (other encoding/ selection/crossover/mutation schemes) and obtain a comparative analysis of the same.
This will be done by the students in the second session.

Tool/Language:
Python

Report Summary:

Problem Solved :
- The goal of the study "Stock price prediction using Genetic Algorithm and Evolution strategies" is to anticipate whether a stock's highest price will rise or fall the next day.
- As a result, it's a binary categorization issue.
- The open, close, high, low, and volume data of a business are utilized as input, and the article attempts to determine the link weight for each characteristic, which aids in the prediction of the stock's target price using the sigmoid function.

Genetic Algorithm(GA) :
- GA implementation begins with a selection of population of chromosome.
- Since we need to find the weights for the 6 attribute, each chromosome will have a 6

numbers indicating the weight. Initial weights are random. The population size is 100

- Data is first normalized and then the weights are used to predict the target price using a sigmoid function. The fitness function is the number of times the weights results in correct prediction
- **Roulette-Wheel Selection :** The selection is done using roulette wheel method where the fitness function acts as the probability of being selected. Thus the chromosome with higher fitness value has a higher chance getting selected and forming off springs.
- **Two-point crossover :** It takes place where in two points are randomly selected in the chromosomes and all the numbers lying in that range are interchanged to produce new off springs. The crossover probability is 0.5
- **Creep mutation** : It is used which adds a small value to each gene with a probability of p. The author has the value of p as 0.013
- Total : 1000 generations.
- The weights obtained after the algorithm stops are used on a test data to determine the accuracy of the algorithm.

Result :

- Training dataset was used to learn the connection weights and then used those weights to test on the test dataset.
- The author observed an accuracy of 73.87% on test dataset which means that 73.87% of the data could predict correctly if the stock price will increase or decrease for the following day

Observation and Conclusion :

- There were a lot of constraints involved in determining the target price: only 6 attributes were used, the algorithm was stopped after 1000 epochs and the weights were limited to the range of 0-1000.
- The author believes that if more varied attributes are used along with some other activation function the accuracy of the algorithm can improve.
- The author concluded that the genetic algorithm performed better than the evolution strategy but believes that with more attributes and enhanced hyperparameters the accuracy can increase even further

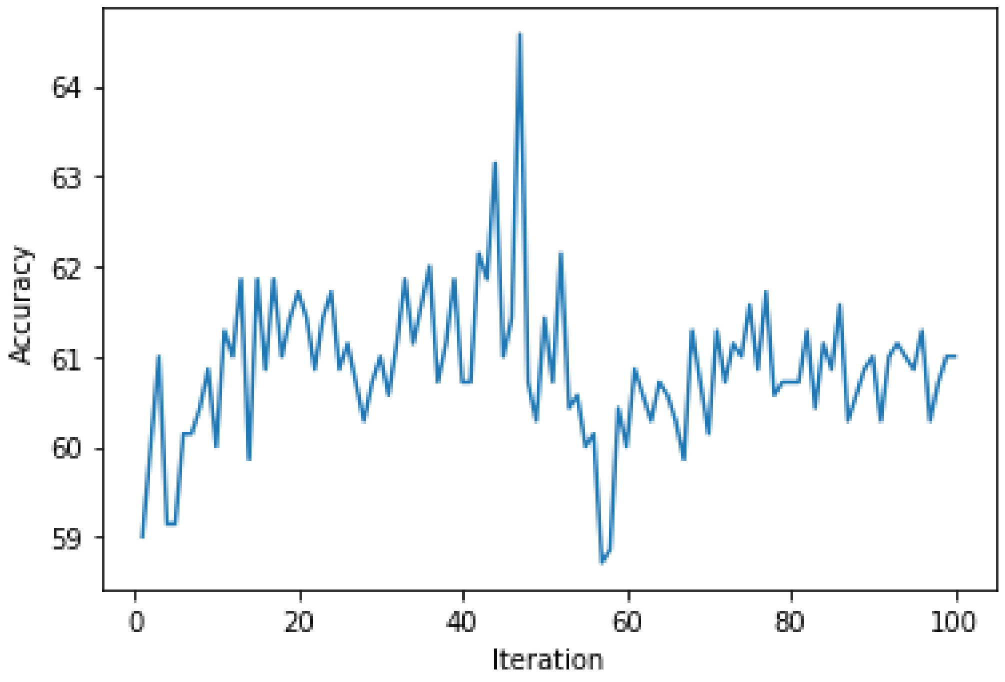
Algorithm :

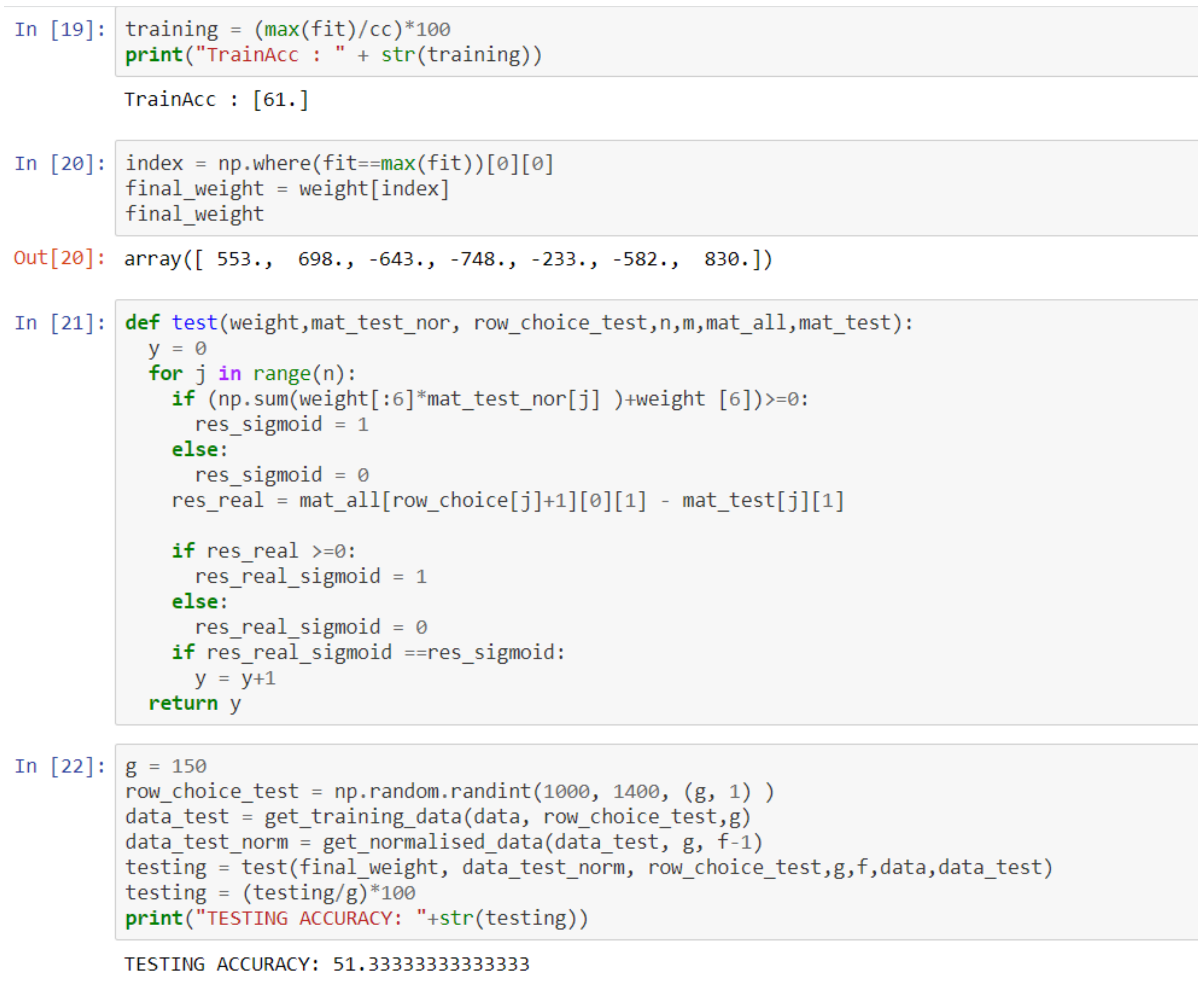
The aim of the paper is to determine the best possible weights for the input attributes that can predict the stock prices correctly. For the same we have used GOOGLE dataset and used genetic algorithm to come up with the best possible weights. This is how the algorithm works :

1. Firstly we take 700 rows of data as training data. Each row has 6 features and thus the shape of training data is (700,6)
2. The training data is then normalized using standard normalization using mean and standard deviation
3. The an initial population of shape (100,7) is created. Each row consists of 7 integers, 6 of which are the weights associated with each input attribute and the last integer acts as the bias.

4. Now for each row in the initial population we calculate the fitness. The fitness of the chromosome is determined by how many predictions can it make correctly
5. From all the fitness value received we get the weights of the best possible fitness
6. Now we implement roulette wheel selection method. In this each chromosome is given a probability according to its fitness function to be selected. Thus the weights with maximum fitness has a higher chance of getting selected
7. Once we get the chromosome we perform two point crossover for each integer in the chromosome. The weights are converted into binary. Two points are generated at random and the bits between those two points are interchanged to get offspring.
8. After this a small mutation probability is set which when satisfied changes one single bit thus avoiding the algorithm from converging quickly. Thus we have got a new set of weights/population
9. Go back to step 5 and repeat the procedure with new weights. This goes on for 1000 iterations.
10. In the end the accuracy of the algorithm is calculated

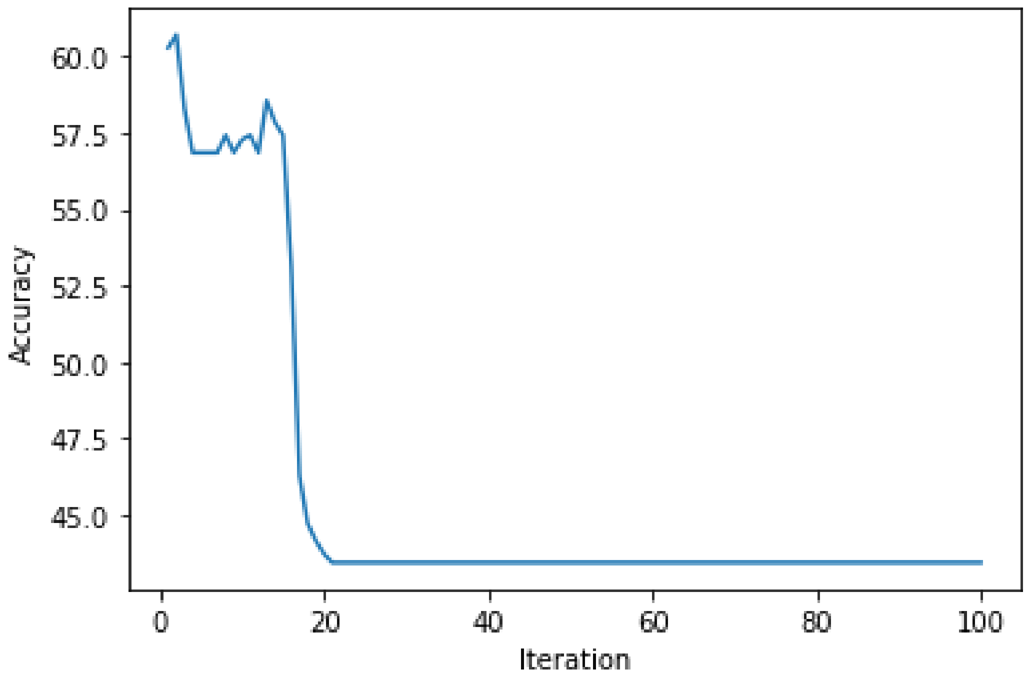
Results:
Two point Crossover





```
In [19]: training = (max(fit)/cc)*100
         print("TrainAcc : " + str(training))

         TrainAcc : [61.]

In [20]: index = np.where(fit==max(fit))[0][0]
         final_weight = weight[index]
         final_weight

Out[20]: array([ 553.,  698., -643., -748., -233., -582.,  830.])

In [21]: def test(weight,mat_test_nor, row_choice_test,n,m,mat_all,mat_test):
           y = 0
           for j in range(n):
             if (np.sum(weight[:6]*mat_test_nor[j] )+weight [6])>=0:
               res_sigmoid = 1
             else:
               res_sigmoid = 0
             res_real = mat_all[row_choice[j]+1][0][1] - mat_test[j][1]

             if res_real >=0:
               res_real_sigmoid = 1
             else:
               res_real_sigmoid = 0
             if res_real_sigmoid ==res_sigmoid:
               y = y+1
           return y

In [22]: g = 150
         row_choice_test = np.random.randint(1000, 1400, (g, 1) )
         data_test = get_training_data(data, row_choice_test,g)
         data_test_norm = get_normalised_data(data_test, g, f-1)
         testing = test(final_weight, data_test_norm, row_choice_test,g,f,data,data_test)
         testing = (testing/g)*100
         print("TESTING ACCURACY: "+str(testing))

         TESTING ACCURACY: 51.33333333333333
```

Uniform Crossover


60.0
57.5
55.0
52.5
50.0
47.5
45.0
Accuracy
0
20
40
60
80
100
Iteration

```
In [34]: training = (max(fit)/cc)*100
         print("TrainAcc : " + str(training))

         TrainAcc : [43.42857143]
```

```
In [35]: index = np.where(fit==max(fit))[0][0]
         final_weight = weight[index]
         final_weight

Out[35]: array([-7.85679815e+64, -1.38034927e+71, -4.16966313e+68, -8.36908890e+70,
                -1.68919265e+56, -2.63280725e+64, -2.14212999e+59])
```

```
In [36]: def test(weight,mat_test_nor, row_choice_test,n,m,mat_all,mat_test):
           y = 0
           for j in range(n):
             if (np.sum(weight[:6]*mat_test_nor[j] )+weight [6])>=0:
               res_sigmoid = 1
             else:
               res_sigmoid = 0
             res_real = mat_all[row_choice[j]+1][0][1] - mat_test[j][1]

             if res_real >=0:
               res_real_sigmoid = 1
             else:
               res_real_sigmoid = 0
             if res_real_sigmoid ==res_sigmoid:
               y = y+1
           return y
```

```
In [37]: g = 150
         row_choice_test = np.random.randint(1000, 1400, (g, 1) )
         data_test = get_training_data(data, row_choice_test,g)
         data_test_norm = get_normalised_data(data_test, g, f-1)
         testing = test(final_weight, data_test_norm, row_choice_test,g,f,data,data_test)
         testing = (testing/g)*100
         print("TESTING ACCURACY: "+str(testing))

         TESTING ACCURACY: 80.66666666666666
```

Accuracy on training and testing data:

Conclusion:

Thus we have implemented genetic algorithm to determine weights for attributes. The training accuracy is of 61% and testing accuracy is of 51.34% for two pointcrossover and for uniform crossover the training accuracy is of 43.43% and testing accuracy is of 80..67% . Genetic algorithm thus can be used in optimization problem and provides a good alternative to other tedious algorithms