INTRODUCTION

This report provides a detailed analysis on the coding work done in fulfilment of COMP307: Introduction to Artificial Intelligence assignment 2. All the coding work for this assignment was done in Python 3 in a Linux Ubuntu 14.04 operating system. The following external libraries have been imported in Python: numpy, pandas, random, re, operator, math, copy, csv and itertools. In addition to this part 1 requires the sklearn machine learning library; part 2 and part 3 requires the deap machine learning library.

PART 1

The objective of this part is to design a neural network to perform classification on the iris dataset provided. The MLPClassifier function in the sklearn python machine learning library is used here.

1. The network architecture consists of four input nodes which are sepal length, sepal width, petal length, petal width and one output node which is class. The class node outputs 2 for Iris-setosa, 4 for Iris-versicolor and 6 for Iris-virginica. The number of hidden nodes used in this architecture is three. This is because it is always advisable to select the number of hidden nodes such that it is less than the number of input nodes but higher than the number of output nodes.

2. The learning rate used here is the default value 0.001. A decrease in the learning rate increases the accuracy of the classifier but also increases the computation time. Since the numbers of data points are less for the iris dataset I was able to use this small learning rate value. The small number of data points available also enabled the use of a large number of iterations i.e 10000 to improve accuracy. The activation function is the sigmoid which brings the value between 0 and 1. The random state value given decides how the train_test_split splits arrays or matrices into random train and test subsets. After some trial and error, the value 42 showed the highest accuracy on average.

3. Since the number of data points were small and the computation time of each iteration was also very small the program was allowed to run for the full 10000 iterations without any early stopping or termination mechanism. The program was able to provide an output in less than a second.

4. Average accuracy on training set for 10 runs=99.48%

   Average accuracy on test set for 10 runs=98.67%

   The results show that the neural network can obtain a very high accuracy for the iris data set provided the parameters are adjusted properly. Both the training and the test set show very good results.

5. When the k-nearest neighbour algorithm was used on the iris-dataset the highest accuracy obtained was at k=1, an accuracy of 94.67%. The k=3 accuracy value was only 92%. But the neural network algorithm was able to get the accuracy rate of 98.67% as mentioned earlier. This spike in accuracy shows that the neural network design works better in this case.

PART 2

The objective of this part is to use genetic programming to evolve a mathematical function for a simple symbolic regression task. The eaSimple algorithm from the python deap machine learning library was used

1. The terminals needed for symbolic regression are an argument x and a random number provided by the addEphemeralConstant function.

2. The function set for this program is addition operator, subtraction operator, multiplication operator, a specially designed division function which does not return an error when dividing by zero, the cosine and sine functions from the math library.

3. The fitness function used here calculates the root mean square error to determine the fitness of an individual. The fitness function calculates the difference between obtained value from a candidate solution and the actual value. The results are stored in a list which is then divided by the number of data points. The obtained value(s) is returned as a tuple because the deap library requies it.

```
def evalSymbReg(x_values, y_values, individual):
    func = toolbox.compile(expr=individual)
    sqerrors=[]
    for i in range(len(x_values)):
        sqerrors.append((func(x_values[i]) - y_values[i])**2)
    return math.fsum(sqerrors) / len(x_values),
```

4. The probability of mating of two individuals was set as 0.5 and the probability of mutation was set as 0.1. It is general practice to give a probability of 0.5 for mating in most programs. The mutation probability should be lower. The program was set to return all the statistics values in numpy arrays. The hall of fame would only contain one individual i.e the best performing candidate solution in the last generation. The program was set to run for 100 generations to provide an accurate solution with a population of 300.

5. With random seed =318
   add(mul(mul(x, add(add(x, mul(mul(x, add(x, mul(mul(x, add(add(x, -1), -1)), cos(x)))), cos(x))), x)), cos(sub(add(mul(mul(x, add(add(x, -1), -1)), cos(sub(add(x, sin(sub(x, 0))), x))), 0), x))), mul(add(x, x), add(add(x, add(x, -1)), -1)))
   Maximum fitness=426.594

   With random seed=310
   add(mul(add(-1, x), add(add(add(x, x), sub(x, mul(x, protectedDiv(cos(x), sin(sin(1)))))), add(-1, x))), mul(add(x, sub(sub(x, 0), mul(add(-1, x), x))), protectedDiv(protectedDiv(cos(x), sin(sin(1))), sin(1))))
   Maximum fitness= 2012.56

   With random seed=325
   sub(mul(add(x, add(x, add(add(add(x, add(mul(cos(x), mul(-1, x)), x)), cos(add(mul(cos(x), mul(-1, x)), x))), cos(add(mul(cos(x), mul(-1, add(cos(add(mul(add(mul(cos(x), mul(-1, x)), x), mul(-1, x)), x)), x))), x))))), add(-1, x)), add(-1, x))
   Maximum fitness= 979.794

Part 3

The objective of this part is to use GP to classify the Wisconsin cancer data set instances into two classes: benign and terminal. The eaSimple algorithm from the python deap machine learning library was used.

1. The terminal set for this task contains the Boolean values true and false; also a random number given via the addEphemeralConstant function in deap.

2. The function set for this task contains the python operators for addition, subtraction and multiplication. The logical operators “and”, “or”, “not”, “less than” and “equal to”. A special division function that does not create an error when dividing by zero and an if_then_else operator is also provided.

3. The fitness function for this task simply checks whether the Boolean output obtained from the candidate solution is equal to the actual Boolean output indicating the result. The sum of the results is returned as tuple.

   ```
   def evalCancer(X_train, Y_train_bool, individual):
       func = toolbox.compile(expr=individual)
       result = sum(bool(func(*X_train[k])) is bool(Y_train_bool[k]) for k in
   range(len(Y_train_bool)))
       return result,
   ```

4. The mating value here is also given as 0.5 since it is general for most tasks. However it was observed that the program got the best results when the mutation probability was 0.2. The statistics for this task was also returned in a numpy array. The number of generations here is limited to 40 since a higher value significantly increases the likelihood of a memory overload due to the unexpected increase in the size of each individual. The population value here is 100. The hall of fame only stores only individual i.e the candidate solution that performed best in the last generation.

5. The data was extracted from the 'breast-cancer-wisconsin.data' file using the pandas library in python. The id number column was instantly dropped since it had no bearing on the likelihood of a person having cancer. All the '?' in the data was replaced by -1 as recommended in the question sheet. The data was then shuffled randomly and 80% was put in the training.txt file and the remaining 20% was put in the test.txt file. The training part requires a lot more data than the testing part.

6. Average accuracy in training set= 0.9785714285714285

   Average accuracy in test set= 0.9496402877697842

7. With random seed=10
   not_(lt(protectedDiv(59.16397313634473, mul(IN6, IN2)), mul(IN4, IN5)))
   Maximum fitness=547

   With random seed=15
   lt(mul(IN5, IN2), protectedDiv(protectedDiv(45.76431326017685, IN6), IN7))
   Maximum fitness=547

   With random seed=5
   or_(and_(and_(lt(sub(mul(IN8, IN1), sub(9.298541960459595, IN1)), IN8), lt(sub(mul(IN1, mul(mul(IN7, IN5), IN5)), IN4), mul(sub(9.298541960459595, IN1), mul(IN1, IN5)))), lt(sub(mul(IN1, IN5), IN4), mul(sub(9.298541960459595, IN1), add(IN1, IN8)))), lt(mul(IN1, add(IN7, IN5)), IN8))
   Maximum fitness=539