Comp307 2020 Assignment 2 Isaac Read

Part 1:

1. Report the classification accuracy your perceptron learning algorithm after running 200 epochs.

The classification accuracy was 0.75 after 200 epochs, this is calculated by using the perceptron to classify all data points and dividing the number of correct classifications divided by the total number of data points.

2. Analyse and describe reasons that your algorithm could not achieve better results.

Although the accuracy was 0.75 after 200 epochs, the accuracy fluxuated from 0.5 to 0.75. This is due to the fact that the data was not linerly seperable, this is a limitiation of perceptrons. They are limited to making classification with linearly seperable data.
While the perceptron had its maximum accuracy of 0.75 and it was presented with a data point which it predicted wrong, it would change the weights values to correct the prediction on that particular data point, this would reduce the overall accuracy. This is why the accuracy fluctuated.

Part 2:

1. Determine and report the network architecture, including the number of input nodes, the number of output nodes, the number of hidden nodes (assume only one hidden layer is used here). Describe the rationale of your choice.

I used Scikit-Learn's implementation of a Multi-Layer Perceptron. I used this package as I am most comfortable in programmning with python and Scikit learn is a well known Machine learning package for python which I have been interested in learning.

I used MLPClassifier because it was designed for this purpose: Predicting labels for data.

For the solver parameter I chose 'lbfgs'. This was recomended by scikit-Learn documentation as the best solver for small data sets. quote - "The default solver ‘adam’ works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, ‘lbfgs’ can converge faster and perform better." I determined the wine data set was small as it had much less then "thousands of training samples".

Inititaly the Network did not perform well, the console logs reported that it was failing to converge and recomemded scaling the data.
I used standard scaler to scale the training and test data around 0 with normal distribution. This ment that each feature was equally weighted, before scaling it would weight the features with higher numbers more.
This improved the results considerably.

The number of input nodes is the number of features plus 1 bias node. 14 total for the wine data.
The number of output nodes is the number of labels so 3 for wine data.

I used wrote my own grid search to detirmine the alpha number and the number of hidden nodes (1 layer) as these 2 parameters seemed to have the most impact. I tried various combinations of alpha ranging from 0.0001 to 100 going up in factors of 10, and number of hidden nodes from 1 to 20. I

actually found that once I hit the sweet spot of 1.0 alpha, adding more then 2 hidden nodes had no impact on average accuracy on the test set. I decided to use the minimum number of hidden nodes providing the best accuracy as this would reduce the complexity of the network, 2 hidden nodes.

2. Determine the learning parameters, including the learning rate, momentum, initial weight ranges, and any other parameters you used. Describe the rationale of your choice.

I did not set any of learning rate, momentum or initial weight ranges. These cannot be set when using 'lbfgs' with Scikit-Learn's MLPClassifier.
You might consider the aplha parameter a learning parameter, as discused above I used a grid seach to determine the best value for this of 1.0 .

3. Determine your network training termination criteria. Describe the rationale of your decision.
The tol paramerter was set at 1e-4. Meaning if the loss did not improve by at least this amount for 2 consecutive itterations the network would be determined to have converged and training will stop. This seemed like a resonable number as any improvments smaller then this would be negligable in terms of increasing accuracy. The max iterations was set at 500, this was large enough that convergance was reached before reaching this max itterations.

4. Report your results (average results of 10 independent experiment runs with different random seeds) on both the training set and the test set. Analyse your results and make your conclusions.

Scored as number of correct classifications divided by total number of classifications.
Average score over 10 independant training and scoring runs with different random seeds:

Training Set: 1.0
Test Set: 0.9775 (4dp)

5. (optional/bonus 5 marks) Compare the performance of this method (neural networks) and the nearest neighbour methods.
The classification accuracy on the test set for my Nearest Neighbour methods was 0.9551 (4dp). My Nueral network implementaion acheived an average of 87 correct out of 89 test instances while the nearist neighbour acheived 85 correct out of 89 test instances. The Nearist Neibour performed remarkably well considering its relative simplicity compared to the nerual network, although the nural network was not particuallay complex with is 2 hidden nodes. I assume this margin of improvment from nearest neibour to neural network would increase as the problem complexity and data size increased

Part 3:

1. Determine a good terminal set for this task.
It depends what you want as a final solution. Firstly, I tried just using random integer constants between 0 and 2 as well as the single variable input, this paied with a minimilistic function set provided simple, understandable programs with quite good accuracy. I then changed the integer constant to a random float from 0 to 1. This sometimes marginally improved the best MSE

individual, allthough sometimes just provided an equivilent or worse program then the simple version.

2. Determine a good function set for this task.
Likewise with the function set; I started simplistic and minimal. Just multiplication, addition, and negation. This paired with the simple terminal set provided simple programs which provided quite good best programs. I then added a 'percent of' function as well as sin and cos functions. These additional functions paired with the float terminal created marginally improved best individual some of the time will introducing alot of complexity to the final program.

3. Construct a good fitness function and describe it using plain language, and mathematical formula (or any other format that can describe the fitness function as accurately as mathematical formula, such as pseudo code).
I summed the squared differences between the prediction of the algorithm and the target with each provided data point and divided the sum by the total number of data points. In other words, the average mean squared error. The program tries to minimise this function
This provides a good fitness function as it somewhat punishes predictions which are close and punishes the predictions which are far off much more harshly.

4. Describe the relevant parameter values and the stopping criteria you used.
I used an initial population size of 500 as recomended by the tutorial. The probability of mating two individuals I set to 0.5. The probability of mutating an individual I set to 0.1. The stopping criteria I used was number of generations, I set this to 60 as it allowed the program to converge completely with each run. I.e The best fitness score ceased to continue improving before the max generations was reached. I did not want to set it higher then this as that would just waste time.

5. Report the mean squared error for each of 10 independent runs with different random seeds, and their average value.
[0.0004001435101109463, 0.0004062287178687352, 0.0004025293302448507, 0.0004000526300839656, 0.0004001173731581354, 0.0004748091777043782, 0.0004054526849302807, 0.000401290077981373, 0.00040153342566421485, 0.0003777250606485226]

avg 0.0004069881988395402

6. List three different best programs and their fitness values.

The best with simplistic termials and functions:

mul(add(2, x), add(x, 2))
MSE Average: 0.00041440

The best 2 I found with more complex terminals and functions:

add(add(mul(x, x), add(add(add(0.9117781610511103, add(add(0.7498377683391291, x), 0.8221762894476989)), x), add(add(percentOf(percentOf(add(add(add(percentOf(percentOf(x, 0.14341055600488728), 0.6615374483438407), x), 0.8221762894476989), x), add(0.9117781610511103, add(x, neg(neg(x))))), 0.6615374483438407), x), 0.8221762894476989))), add(x, 0.6857709117434835))
MSE Average: 0.00039187

add(add(add(add(x, 0.5977502148819956), add(add(0.8700477649320717, 0.9862084407406473), percentOf(0.05048108097535764, add(add(x, sin(x)), add(x, 0.5977502148819956))))), add(add(add(add(x, 0.5584285715916343), 0.9862084407406473), x), x)), mul(x, x))
MSE Average: 0.00038912

7. (optional, bonus, 5 marks) Analyse one of the best programs and explain why it can solve the problem in the task
Take mul(add(2, x), add(x, 2)). This program can solve the problem because when you input an x value from the training set into this function and do the prescribed calculation, it provides a close approximation for the corrasponding target y value. Sometimes it is completly accurate sometimes it is a small amount off, depending on which x you input. This is why it can solve the problem in the task.

part 4:

1. Determine a good terminal set for this task.
I used the same technique as in the last part, initailly just using integers from 0 to 2 and the set of 36 variables as potential nodes. And then trying a random float from 0 to 1 instead of integers. However it seems the best programs never use constansts at all, only using the variabls and functions. This is perhaps because of the nature of this data set and the fact that i am using 0 as a threshold value for classification.

2. Determine a good function set for this task.
As with the last part. I used just add, multiply and negate intitially. This provided simple but good programs. I tried adding Sin and cos but when ever these were used in the best program the error rate was higher on the test set. I added absolute function and percent of, these provided marginal gains on the more simplistic approch. I did not use minus function as this is equivilent to negation and addition. I did not use divide as this is somewhat equivilent to percent of with the added problem of divide by zero. I did not use exponential as I ran into overflow issues.

3. Construct a good fitness function and describe it using plain language, and mathematical formula (or any other format that can describe the fitness function as accurately as mathematical formula, such as pseudo code).

Because it was a classification problem, converted the number output of the program to a classification by using the threshold of 0. If it was greater then or equal to 0 it was deemed normal, else anomaly.

I initially tried punishing the program more the prediction was further away from the truth. This was not as successfull as just a simple approch of summing the number of incorrect guesses and dividing by the total number of guesses. The program tried to minimise this function.

4. Describe the relevant parameter values and the stopping criteria you used.
I used the same parameter values and stopping criteria as last time. Initial population size of 500 as recomended by the tutorial. The probability of mating two individuals I set to 0.5. The probability of mutating an individual I set to 0.1. The stopping criteria I used was number of generations, I set this to 60 as it allowed the program to converge completely with each run. I.e The best fitness score ceased to continue improving before the max generations was reached. I did not want to set it higher then this as that would just waste time.

5. Describe your main considerations in splitting the original data set into a training set training.txt and a test set test.txt.
I wanted to maintain the same ratio of normal : anomalys in the test and training set. This means that the program would be properly judged. If this ratio was not maintianed the accuracy on the test set could be inflated or deflated if there were some bias toward normal or anomaly.

I used a 0.2 * total for the size of the test set. The remaining was training set. I wanted to have a large enough data to train properly on while still having enough tests to accuratly assess the programs succsess.

6. Report the classification accuracy (average accuracy over 10 independent experiment runs with different random seeds) on both the training set and the test set.

Average Test set accuracy: 0.8263 (4dp)
Average Training set accuracy: 0.9374 (4dp)

7. List three best programs evolved by GP and the fitness value of them.

Simpler terminals and functions set:

add(ARG17, neg(ARG3))
TestAcc = 0.7894736842105263
Fitness value = 0.083871

More complex Terminals and functions set:

add(add(add(neg(ARG27), add(add(ARG6, add(neg(ARG23), percentOf(ARG34, ARG13))), add(add(neg(ARG15), add(add(ARG6, add(neg(ARG23), percentOf(ARG34, ARG13))), add(neg(abs(ARG4)), add(neg(ARG7), percentOf(ARG9, ARG34))))), add(neg(ARG7), percentOf(ARG9, ARG34))))), add(ARG13, add(neg(abs(ARG4)), add(neg(ARG7), percentOf(ARG7, ARG34))))), ARG17)
TestAcc = 0.9474 (4dp)
Fitness value = 0.0258065

add(neg(ARG3), percentOf(percentOf(ARG5, ARG21), add(percentOf(percentOf(ARG5, ARG21), add(ARG33, ARG35)), ARG21)))
TestAcc = 0.9210526315789473
Fitness value = 0.0322581

8. (optional, bonus, 5 marks) Analyse one of best programs to identify patterns you can find in the evolved/learned program and why it can solve the problem well (or badly).

Take add(ARG17, neg(ARG3)).
This solves the problem well because it seems generally if you take arg3 (V4) away from arg17 (V18) and this is above or equal to zero it is a good sign that it is normal, as aposed to anomaly data. So if V4 is smaller then V18 then its a good predictor of normal data.