CSC 503

Assignment-1

Data Mining Analysis Report

Submitted to - Dr Nishant Mehta

Submitted by – Divyansh Bhardwaj

V# - (V00949736)

INTRODUCTION

This is an analysis report based on the performance of different Machine Learning Models implemented in Python. The problem given to us was a binary classification problem, based on the whether the patient under consideration has some kind of heart related disease or not. The dataset was provided on one of the repositories prepared by the UCI (University of California, Irvine). The dataset was further cleaned and provided to us by our Instructor of this course, Dr Nishant Mehta.

The dataset has 13 features and a target variable (binary classification) which takes the value either 0 or 1 (0 indicating the absence of heart disease and 1 indicating the presence of the heart disease). The dataset contains 297 samples or records of patients which are analysed and worked on.

The other problem was expected to be created by us where we had to choose one of the datasets or create one, which again had some interesting classification problem. After searching for a lot for the datasets for the same, I realised that it will be much better to create the dataset myself. There are several libraries in python which let you create your dataset for classification as well as regression problems. But the problem had to be interesting, so I thought of something in my mind while creating that particular type of dataset. The reason for which I chose to have that type of dataset will be explained in the upcoming pages of the report. After doing my analysis of the first dataset which was already provided to us, it was clear as to why should one go with that dataset. That will be discussed soon, but before let us have a look at the analysis of the heart disease dataset.

ANALYSIS ON THE HEART DISEASE DATASET

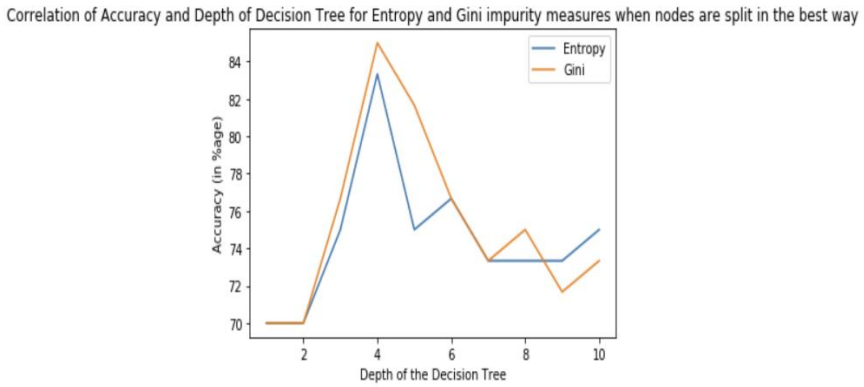
1.) Decision Tree

The Heart disease dataset was divided into testing and training datasets. The ratio was kept according to the rule of thumb i.e. 80:20 (80% of the original is training data and the rest 20% is the test data). The Decision tree implementation was done in python. A library called Sci-Kit learn (commonly referred as sklearn) was used to instantiate the decision tree. There are certain parameters which could be tuned in to get a better accuracy or performance of the decision tree. Below are mentioned various analysis that were done on the decision tree.

The analysis that are first shown below are the one's where the decision tree has not been pruned in any way.

- **The effect of depth of the tree on ACCURACY when the tree is trained using both the impurity measures (GINI and ENTROPY) and the nodes are split using the BEST splitter:**

```
maximum accuracy achieved with Entropy is :
83.33333333333334
maximum accuracy achieved with Gini is :
85.0
```

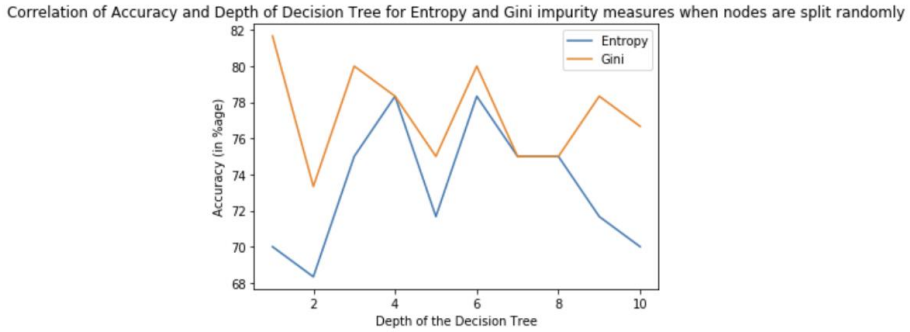


ANALYSIS: There are two ways to split a node 'best' and 'random'. If the nodes are split in the best way, that means the best feature will be considered to split the node. If the splitter parameter is chosen to be 'random' then it chooses a random subset of features to split the node. **Here the analysis is done for the best split.**

So, for the analysis with the depth parameter, I increased the depth of the tree from one to 10 and trained the decision trees for different depth once with the **Gini index** and then with the **information gain**. It turned out that Gini index tends to give a better accuracy everytime as the curve for Gini index tends to go steeper than the curve for entropy. Though the difference between the accuracy is not much, but it was concluded that Gini Index tends to perform better than the information gain.
The maximum Testing accuracy achieved by the Gini index was 85 % and that of the Entropy was 83.34 % (approx).
It should be kept in mind that no pruning has been done on this decision tree.

- **The effect of depth of the tree on ACCURACY when the tree is trained using both the impurity measures (GINI and ENTROPY) and the nodes are split using the RANDOM splitter:**

```
maximum accuracy achieved with Entropy is :
78.33333333333333
maximum accuracy achieved with Gini is :
81.66666666666667
```



ANALYSIS: When the nodes are split using some random subset of features, the overall accuracy of the decision tree is decreased. Yet the Gini Index achieves better accuracy in this case as well. Th curve for Gini Index tends to stay steeper as can be clearly seen from the picture.
The maximum accuracy achieved by decision tree trained on Gini index is 81.67 % whereas it was 78.34 % in case of entropy impurity measure.
This is again the case where the pruning has not been done for the decision tree.

- **Accuracy shown by dividing the dataset into <u>Testing and Validation</u> and then applying <u>Post Pruning</u> to it.**
  The post pruning is applied to the decision tree by optimizing the cost complexity measure which is given by :-

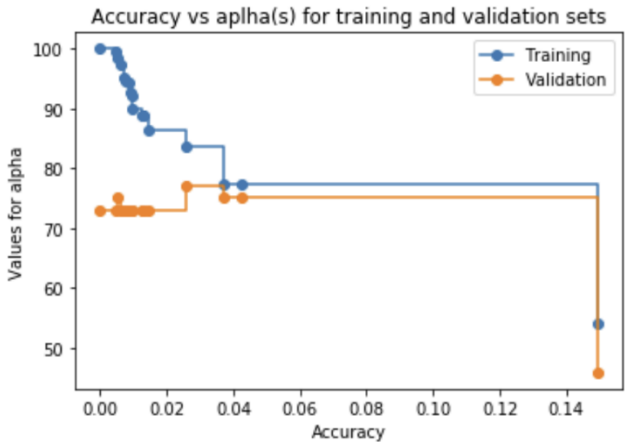
$$R_\alpha(T) = R(T) + \alpha|T|$$

R(T) — Total training error of leaf nodes
|T| — The number of leaf nodes
α — complexity parameter

First the tree is grown into a fully complex tree with no pruning done at all. Then for every node, the effective/optimized values of alpha are considered which tells you which node to be considered. Greater the value of alpha means more nodes are being pruned. So we compute all the values of alpha and choose the optimized value for it.

So I drew a graph where values of alpha on the x axis are plotted against the accuracy on y-axis to check how the values of alpha affect the accuracy for both testing and validation data.

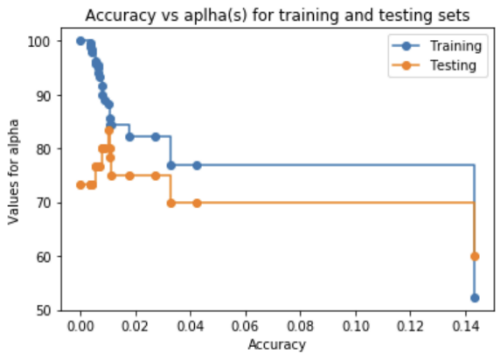
```
Out[12]: Text(0, 0.5, 'Values for alpha')
```



ANALYSIS: It can be clearly inferred from the graph, that the for increasing values of alpha the testing accuracy decreases, and the validation accuracy increases to a certain point. Then the accuracy remains constant for a certain amount of time and then both the accuracies start to fall down. This is how the validation testing can act as a proxy for the training accuracy. And the overfitting is combatted.
The validation accuracy turned out to be 78.89 % and the value of alpha was obtained to be around 0.025 (approx).
(Note: The axes have been labelled incorrectly, Accuracy is on the y axis and the values for alpha are there on x axis. Apologies for the inconvenience. Due to time constraints this could not be edited.)

- **Comparison of Accuracy between the Training and Testing datasets after applying the Post Pruning to the decision tree.**

  The graph that is shown below depicts the values for alpha against the accuracy of both training and testing datasets.
  (Note: Similar mistake has been done here also. The labelling of axes is wrong. Accuracy is meant to be written on the y axis and values for alpha on the x axis. Inconvenience is regretted.)
  After looking at the above graph for training and validation datasets, we got to know the final decision tree which could now be used to calculate the testing accuracy for the same. In other words, we got to know the value for alpha for which the pruning will be done by the parameter '*ccp_alpha'*. Below shown is the graph for testing and training accuracy.



Analysis: From the above graph, we can clearly see that the overfitting is being combatted at first. The training accuracy is taking a steep fall whereas for that period the testing accuracy is increasing. They come close to a point and then start decreasing.
I computed the maximum testing accuracy and the value for alpha for which the decision is pruned to its best.
Now we will compare the accuracy obtained for the decision tree when **no pruning** was done and **even depth was not tuned**. Below are the results:

```
Accuracy score for Entropy:
75.0
Accuracy score for Gini:
73.33333333333333
```

Now we look at the results, where the tree was pruned and we also print the value for alpha for which the tree is performing at its best.

```
The testing accuracy after pruning (using GINI index) went upto:
83.33333333333334
The optimal value of alpha is
0.010046212577858147
```

We get the accuracy from 73 % to 83.34 % which is a huge increase. Hence, we get some amazing results here.

2.) **<u>RANDOM FORESTS:</u>**

Random forest model was implemented on the same dataset (heart disease). Random forests is a collection of decision trees. It basically works on the principle “wisdom of the crowd”. The majority vote is taken by the random forests and deliver a decision accordingly. There were several hyperparameters of the Random Forest model which could be tuned in for optimizing the performance. The parameters which I tuned in here were as follows:

- Number of trees
- Bootstrap (randomly sampling, with replacement)
- Number of features used
- Number of maximum samples used

- **Performance of the Random Forest Classifier:**
  The performance of the random forest classifier was assessed with all the default parameters in place. Below are the results:

Analysis: As expected the performance of the Random forest classifier with no pruning is much better than the decision tree which was not pruned.

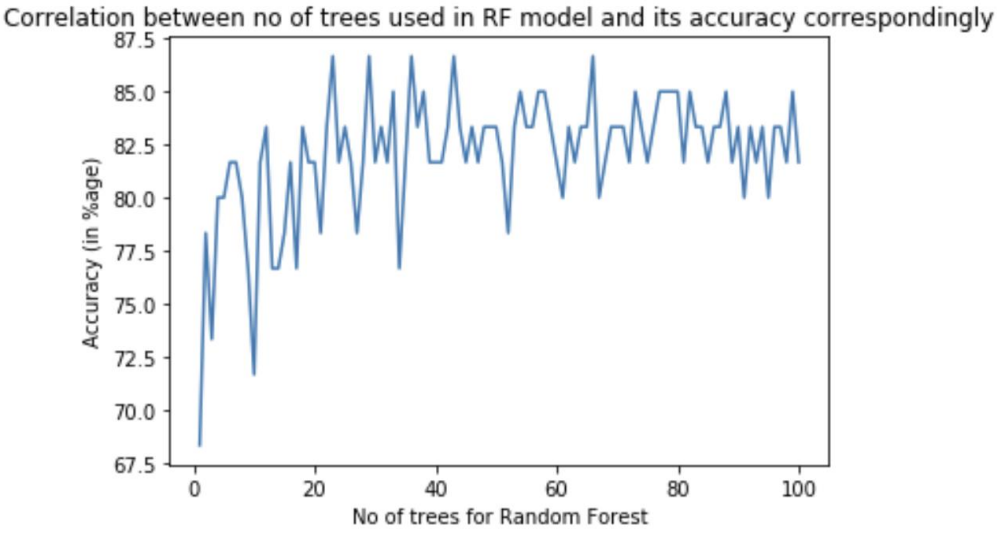
```
In [18]: label_predict = predict_rf(feat_train, label_train, feat_test)
         acc = comp_accuracy(label_test, label_predict)
         print('Test Accuracy for the Random Forest model is :')
         print(acc)

         Test Accuracy for the Random Forest model is :
         83.33333333333334
```

The accuracy came out to be 83.34 %. But the major challenge with the random forest classifier is that which hyperparameters should be used such that the performance is optimized. So I drew the graph of accuracy of classifier with several hyperparameters to see the effect in the accuracy. Below shown analysis are all based on that.

- **The correlation of Accuracy with the Number of trees used in the Random Forest classifier.**
  The accuracy was evaluated by training the model again and again with the changing values of parameter to get the following results.

Analysis: As the number of trees went on increasing to a certain limit, the accuracy went on increasing. But after a certain amount of trees used, the accuracy curve flattened. Below shown are the results:

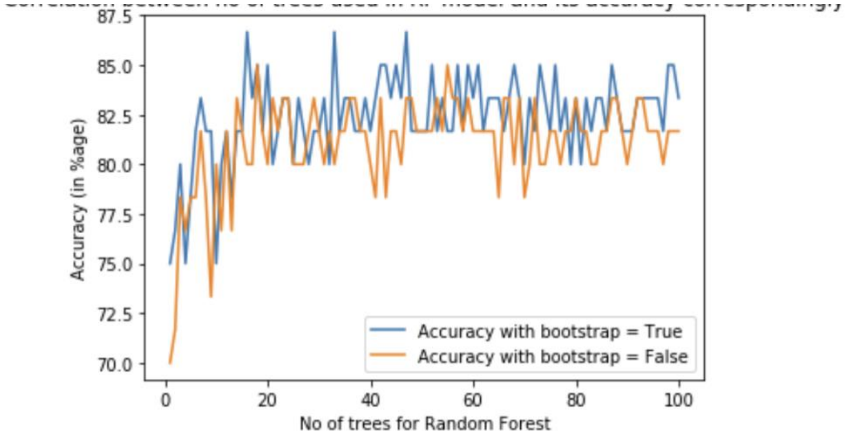


The first thing that should be noted is the nature of the curve. The values for accuracy are fluctuating constantly going up and down like a polygraph of ecg curve. This proves the randomness of the model. Everytime the model is trained, we get different values for the model.

Secondly, and most importantly, here the after the no of trees reach more than 20, the accuracy is randomly fluctuating around a region and the curve becomes constant. In the range 20-40, it achieves the maximum value. Mean comes around 83 %.

- **Comparison of accuracy when tuning the parameter "Bootstrap"**
  Bootstrap parameter takes two values 'True' or 'False'. True value indicates that the samples are being randomly sampled with replacement. If the Bootstrap parameter is set to false, the whole dataset is used to build the tree. So I compared the affect on accuracy when the bootstrap parameter is toggled between true and false.

Analysis: The parameter when toggled between true and false shows a very slight difference between the accuracy of the classifier. The curve for accuracy with bootstarp = true always tends to have a higher accuracy than setting the parameter to false. The x axis for the plot was set to be the no of trees. Shown below are the results:

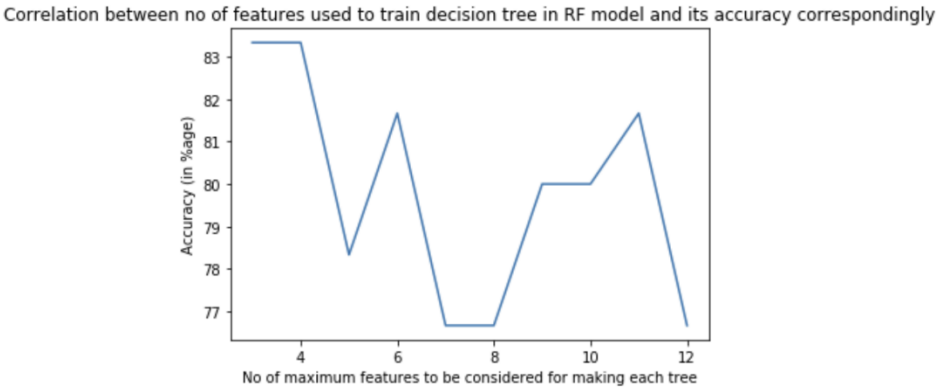


As is clear from the figure, the Accuracy with setting the bootstrap parameter to True is slightly greater than the one with the false value. The mean value for bootstrap = True came out to be around 83.5 % whereas with bootstrap = false, the mean value for accuracy came out to be 82.22 %.

- **Comparison of accuracy when tuning the parameter "Maximum Features used to split the node"**

  The total number of features used in the dataset were 13. Hence I checked on what value of the parameter, the random forest classifier gives the best result. I looked for an interesting result here which is sort of an assumption but as this an analysis report, so we are allowed to put some assumptions the report too to verify whether they are correct or not. The results are shown on the next page.

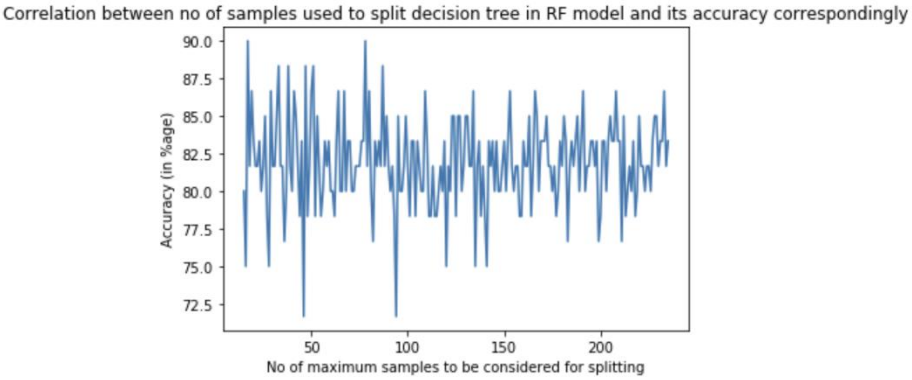
Analysis: It seemed best to plot a graph for this and see what effect does it causes on the accuracy. Shown below are the results.



The maximum value of the accuracy comes when the maximum features roam around the value 4 (<4) in this case. Now there is an interesting fact to look on in this parameter. The default value of the parameter is the taking the (no of features in the dataset)$^{1/2}$. One more value which is provided in the set of values is the $\log_2$(no of features in the dataset). Now there were total of 14 features in the dataset. Taking square root of 14 yields 3.7 (approx). Taking $\log_2(14)$ yields 3.8 and all these values are around 4. Hence it is an assumption of mine that these values are not randomly provided by the library implementing RF model. These values actually yield the best accuracy and hence they are there in the values of parameter.

- **Comparison of Accuracy when tuning the parameter number of samples used.**
  The last parameter that I wanted to tune in was the Number of samples used. The total number of samples in the dataset were 297. Hence I tried adjusting the value of parameter till 297.

Analysis: The graph for accuracy vs number of samples used is shown below:



I tried tuning in the values starting from square root of the number of samples used till the total amount of samples there in the dataset. The value for accuracy slightly better in the range of values [40,70] which is again the values near the square root of the total samples. But overall there is not a significant difference when tuning this parameter. Hence it can be ignored.

3.) Neural Networks:

The same dataset was trained by the neural networks to do my analysis. The parameter which I tried to tune in were

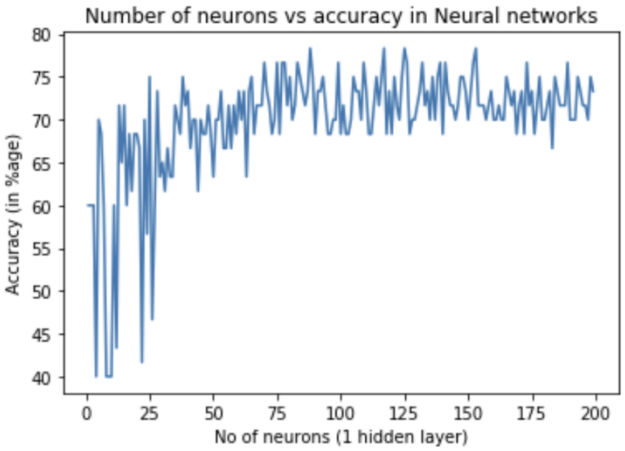
- Number of neurons used
- Number of iterations
- Number of Hidden Layers

- **Comparing the performance of the neural network with changing the neurons in the hidden layer**

  For simplicity, I considered just one hidden layer and tried to check how the neural network performed.

Analysis: With the neurons increasing, the performance of the neural network also increased. Below is the graph shown:

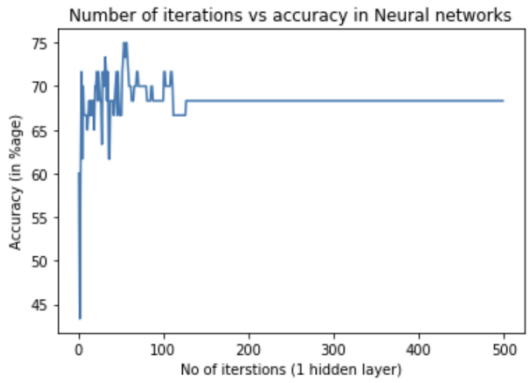
```
Out[172]: Text(0.5, 1.0, 'Number of neurons vs accuracy in Neural networks')
```



It is quite clear from the graph that the accuracy is increasing with the increase in the number of neurons. But the accuracy becomes constant even after increasing the number of neurons. From the graph we can see that this number is around 80. The mean value of accuracy tends to be constant after this number, which is around 73.5%

- **Comparing the performance of the neural network with changing the number of iterations (epochs) in the hidden layer**
  The number of iterations parameter was judged, and correlation was found among the two.

Analysis: The accuracy tends to increase a bit first and then flattens out completely. Hence after a certain point there is absolutely no variation in accuracy after increasing the number of iterations or epochs. I took the number of iterations from 1 to 500. The graph below shows the results.



As is seen from the graph, the number if iterations depends on the amount of data we have. For this small dataset, the number of iterations or epochs necessary to do the optimization is around 120. After that, it's a flat line.

- **Changing the hidden layers of the Neural network**

Analysis: The number of hidden layers were changed to see any change in the accuracy gained by the multi layer perceptron. Since the data was quite small, 1 hidden layer was sufficient to do the stuff. So changing the hidden layers did not bring any kind of improvement in the performance of the neural network classifier.

Overall Analysis on the Heart Disease Dataset

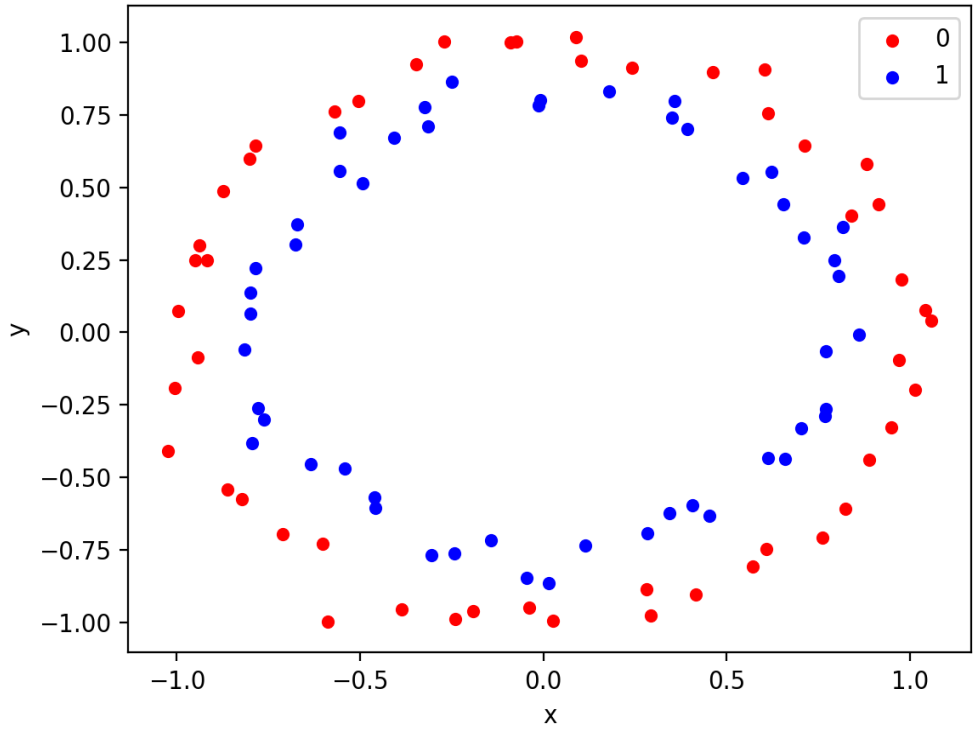
The best performance among all the three models implemented was actually a tie between the two models, Decision Tree (After Reduced error pruning) and Random Forest. The decision tree when post pruned gave the accuracy of 83.34 % which was exactly the same accuracy achieved by the Random Forest Model. Neural networks performed poorly on this dataset.
Talking about Decision trees and Random forests, I was assuming random forests to outperform decision trees. The reason for Random forests not performing better than decision trees , according to my analysis, is that the trees used in building the random forests were not optimized. We did not apply any kind of pruning to the decision trees. Nor did we tune any parameters (like depth) of decision trees which were used to construct the random forests. If we would have done that, the performance of Random forests would have been much better than what it is now.
The reason for neural networks to perform bad on this dataset, according to my analysis, is the lack of data. The deep learning networks are always hungry of the data. Their performance relies on the amount of data that we feed them. This dataset was too small to get the proper performance of the neural network. Secondly, neural networks is a black box. One never knows what really is going on in the hidden layer. Hence to be concrete in what we are saying becomes a bit difficult in this scenario.

Important Conclusion: From what we have seen above, we have to rely on the hit and trial heuristic to analyze the parameter or the hyper parameters for the model which is not a good approach to do. Instead, for machine learning models, which rely on hit and trial the most for tuning in the hyperparameters, we should try something called **Cross Validation.** There are several techniques that

does this tuning of the parameters and give us the best values for the parameters which would result in the most optimized performance. One such technique which can be used is the K-Fold cross validation.

Analysis performed on the Self created Dataset

First I will be specifying the **reason for making that dataset.** After looking at my previous results, I wanted to create an interesting classification problem where the results will be a bit opposite of what they have been till now. With this thought in my mind, I went on to creating a dataset which was highly complex in terms of the classification problems. The dataset has been created by a function called make_circles, in sklearn library of python which creates a dataset as shown below:-



The dataset comprises of 2 dimensional points in the form of circles. The total number of samples produced are 1000. So in short, we have 1000 examples of 2 dimensional points. As is clear from the picture, the dataset is binary classification problem with a fairly complex decision boundary, which will be a circle too. The basic thought behind creating such dataset was to show the importance of Neural networks, so that we get to know where actually we can find the real application of neural networks. But the results came out of the blue! Let us discuss about the performance of the models which were implemented.

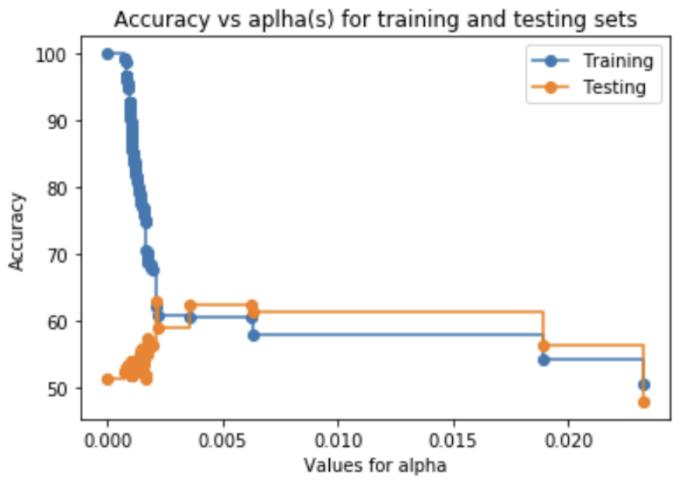
The basic idea was first to get all the values for hyperparameters or maybe a good estimate, so that we can apply it onto the next dataset. So there were several results of the previous analysis, which have been used in this analysis as well.

1.) Decision Tree:

While implementing the decision tree, I took various generalisations from the previous analysis and applied them on the newer dataset to see how it performs under the best conditions. So from the previous analysis, we know that Gini Index performs better than the Entropy. Moreover, we also saw that it is better to do the post pruning of the decision tree so that we get the best performance of the decision tree. Based on these conclusions I started doing my analysis.

- **Checking the performance by calculating the best alpha for which the decision tree is optimized (Post pruning)**

Analysis: The performance of the decision tree, as it was expected, came out to be quite low in comparison to the previous dataset. Even after optimizing the value of alpha, the accuracy came out to be around 63%. The results are shown below:



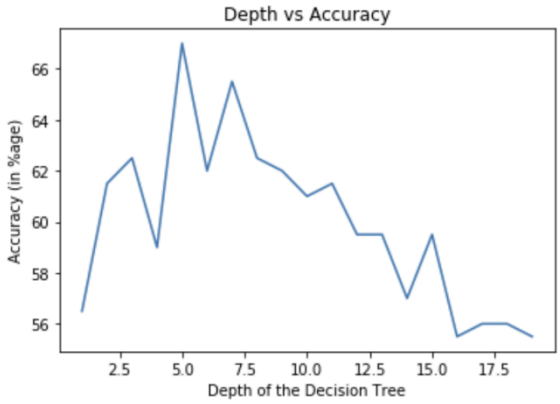
The highest testing accuracy which was achieved by the decision tree classifier was 63.0%. Due to the complexity of data, the decision tree classifier performed a bit poorly.

- **Checking the performance of decision tree by adjusting the depth of the decision tree**

Analysis: As seen from the previous analysis, the performance of the decision tree is quite sensitive to the depth of the decision tree. The results are shown below:

```
Maximum accuracy achieved by tuning the depth is :
67.0
```

```
Out[62]: Text(0.5, 1.0, 'Depth vs Accuracy')
```



This time the maximum accuracy went up to 67 % by adjusting the depth of the tree. In the previous analysis also, it touched around 85%. Hence, the performance of the decision tree is quite sensitive to the depth parameter. Here, for this dataset, the optimal depth came out to be a around 5.

2.) Random Forests:

Random forest classifier was then implemented on the highly complex dataset. The expectations from the model were a bit on a higher side. From the previous analysis, we know that the number of features used to split on should be the square root of the total number of features present in the dataset. Hence in the analysis below, we will be training the model accordingly. The analysis is done below.

- **Accuracy achieved by the Random Forest Classifier**

Analysis: The Random forest classifier was expected to give accuracy on a bit higher side than the decision tree. But the analysis turned out to be the other way. Below is the result shown for the same:

```
Test Accuracy for the Random Forest model is :
59.0
```

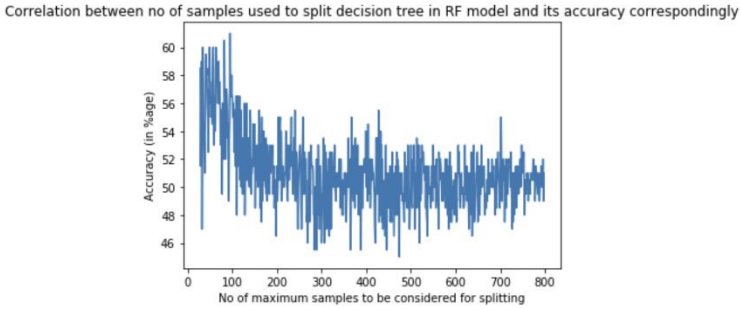
```
In [108]: print(comp_train_accuracy(train_feat, train_label))
```

```
98.75
```

The test accuracy for the random forest classifier came out to be as low as 59 %. So, I thought about this for quite long. The reason that struck me is that the trees used in the random forest model has not been optimized. As we have already seen, after tuning the parameters like depth and doing the post pruning, we get much better accuracy for decision trees. Hence the poor

performance of random forests is because the decision trees used to make the random forest are themselves not optimized.

- **Adjusting the number of samples and checking the accuracy**
  One thing that we noticed in the previous analysis is that in case of random forests, the accuracy of the classifier was not affected by changing the hyper parameter number of samples used to make the tree. Hence I checked the parameter again to see, if there is any scope of generalisation that I could form. Turns out that something was there hidden in the analysis. Below are the results shown.




```
In [115]: print('Maximum accuracy achieved by Random Forests in this dataset is:')
          print(max(acc))

Maximum accuracy achieved by Random Forests in this dataset is:
61.0
```

By tuning in the parameter, the accuracy of the model got improved. It went upto 61 %. The most important thing to note is that as we increase the number of samples to make the decision tree, the accuracy falls down. Accuracy was the highest when the number of samples were below 100. As the number of samples increased, the accuracy started going down steeply.

3.) Neural Networks:

From our previous analysis of the neural networks, we saw that tuning some of the parameters like number of maximum iterations (epochs) and adjusting the number of neuron in the multi layer perceptron yields better results. Hence I altered them here and there to see what are the optimal values for these parameters. Moreover, for this particular case, the neural networks were mean to perform much better than the other models. But the results were totally unexpected.

- **Performance of the Neural network.**

Analysis: The performance shown by neural network for this dataset was one of the lowest. It came out to be around 55%.

```
test_label_predict, train_label_predict = predict_nn(train_feat, train_label, test_feat)
```

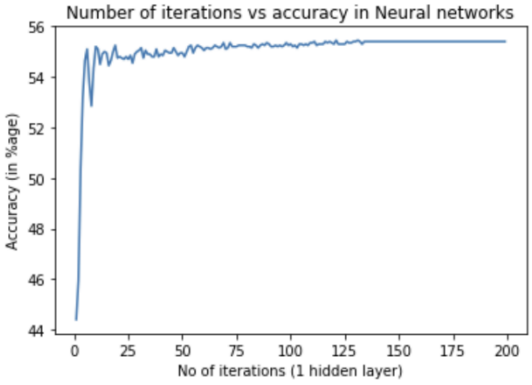
```
print(comp_accuracy(test_label, test_label_predict))
```

```
55.35
```

Neural networks were mean to perform much better than the other models in this dataset. But the results were totally unexpected. There can be a few possible reasons for this thing happening. As discussed earlier, neural networks are always hungry for more and more data. This dataset comprised of a total of 1000 samples and was 2 dimensional. So the first possible reason, according to me, for such a result is the less amount of data available for the networks. The second possible reason for this is, that here I have implemented one of the already existing implementations of the neural networks. Since this is my first time experimenting with it, I might have given some wrong parameters, or maybe somewhere something in the coding part was wrong.

- **Comparison of Accuracy with the number of Iterations performed(epochs)**

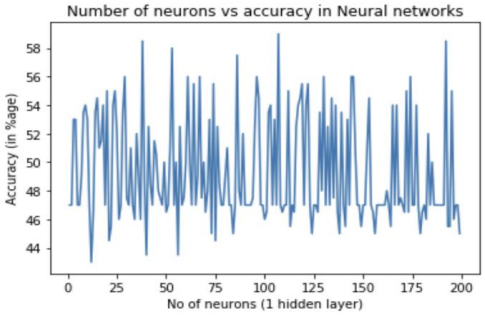
Analysis: As discussed earlier also, the accuracy increases with the number of iterations performed to train the neural networks. Below are shown the results:



As is clear from the above graph, the accuracy is increasing sharply with increasing the number of iterations for this dataset. But the thing that is not understandable is after a very few epochs the accuracy stops increasing for the dataset. After almost 120 iterations, the accuracy is flattened.

- **Comparison of the accuracy with the number of neurons**

Analysis: Below are shown the results.



With the increase in the number of neurons, the accuracy is neither increasing nor decreasing. For neural networks, the analysis for the datasets is coming out to be totally unpredictable. Either there is a problem in my implementation, or the understanding of the parameters involved for the same, or it's the less amount of data available that is causing the neural networks to perform poorly.