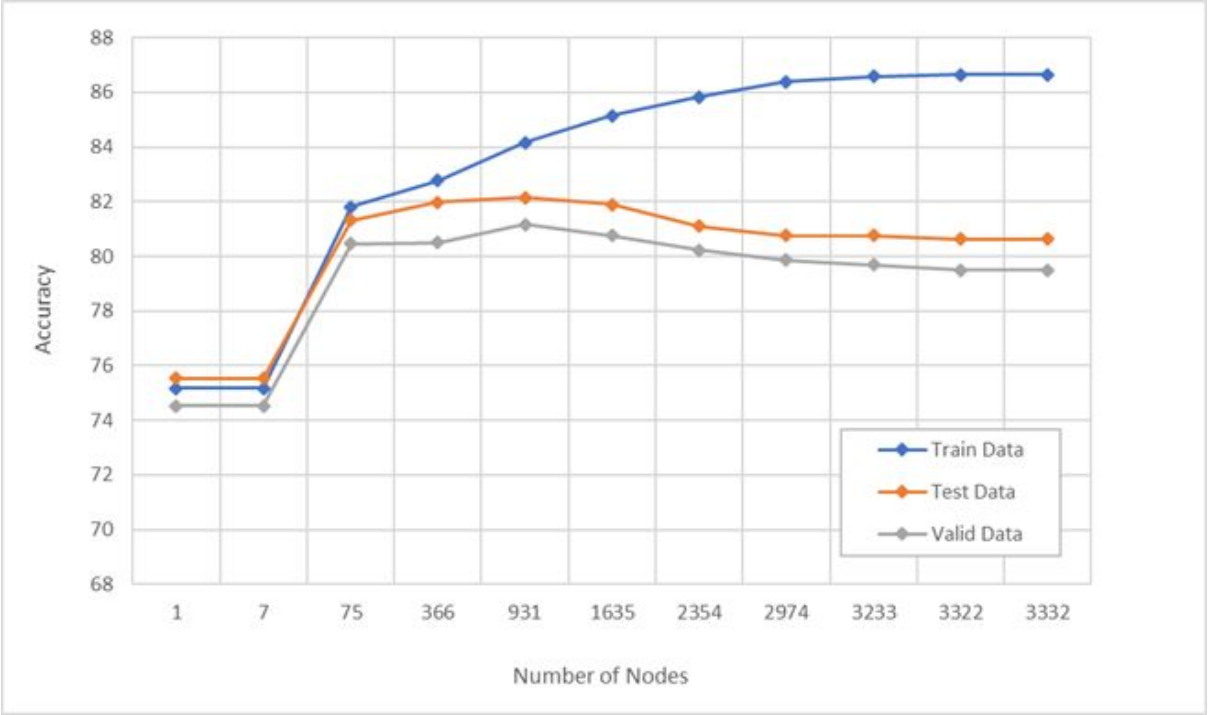
ASSIGNMENT-3

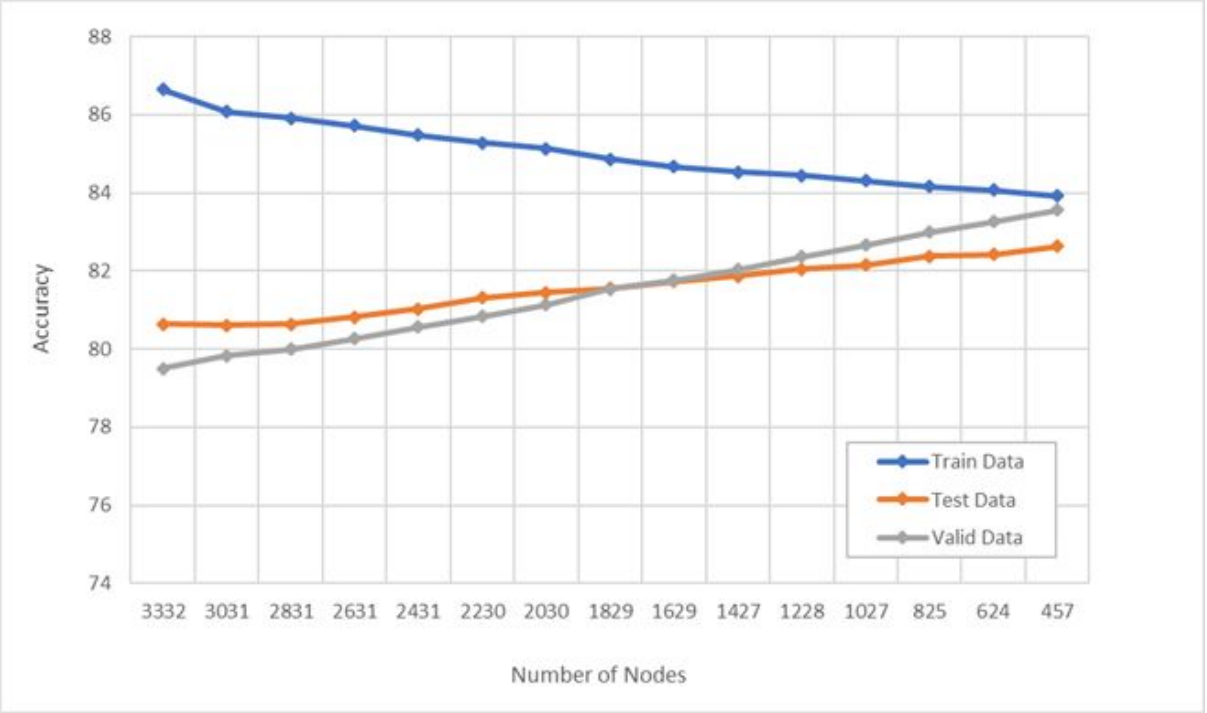
1) **Decision Trees**

(a) Train accuracy: 86.65%
Test accuracy: 80.64%
Valid accuracy: 79.5%



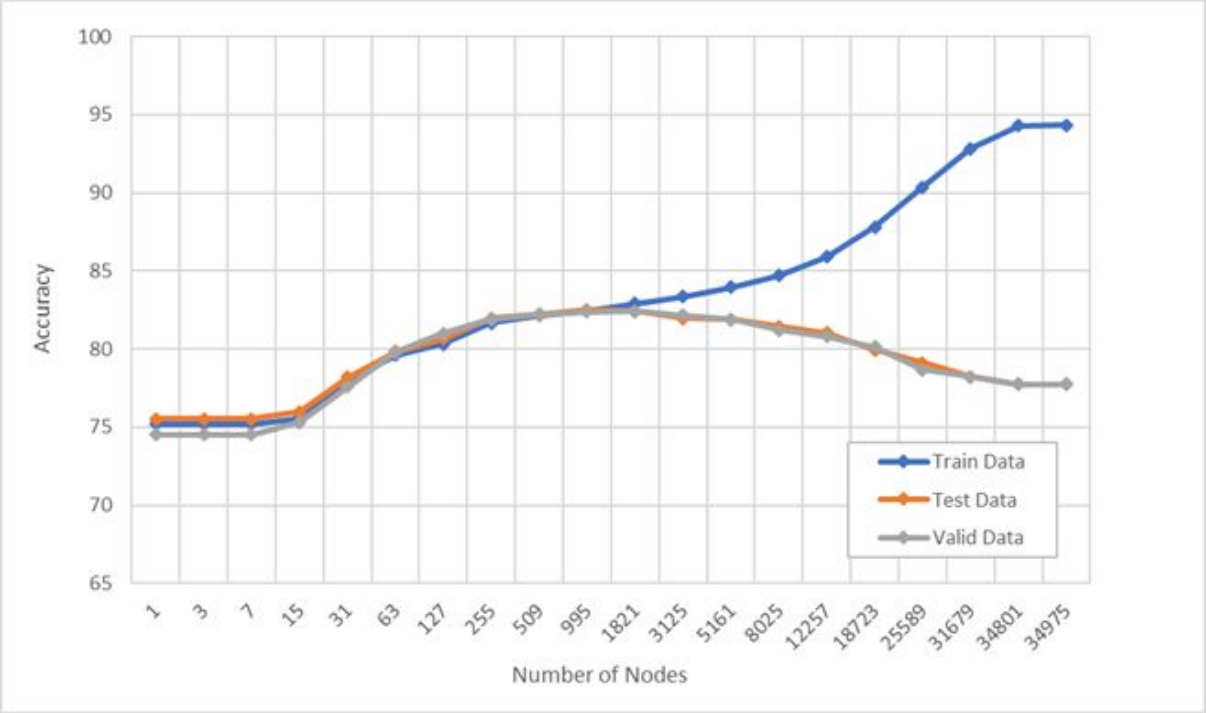
We see from the graph that as the number of nodes increase, the training accuracy continuously increases, that means with increase in the number of nodes, our graph fits the training data more and more and hence the graph is able to predict labels on the training data more and more accurately.
On the other hand, the accuracy for test data and validation data increase with the increase in the number of nodes initially. The for test data, at somewhere around 300-900 number of nodes, the accuracy reaches it's maximum and henceforth starts to decrease. The validation data also follows a similar trend.
This means that with increase in number of nodes, the tree starts overfitting the training data due to which the accuracies on train and validation data are compromised. All in all, the decision trees exhibit good accuracies for new data.

(b) Train accuracy: 83.92%
Test accuracy: 82.62%
Valid accuracy: 83.56%



We see from the graph that as the pruning of nodes start from 3332 and as the number of nodes start to decrease, the training accuracy continuously decreases, that means with decrease in the number of nodes, our graph stops overfitting the training data like it does in part (a).
On the other hand, the accuracy for test data and validation data increase with the decrease in the number of nodes. The optimal accuracy is reached at 457 number of nodes where the accuracy for training data and validation set are almost close and the accuracy for the test data is better than before.
This means that in post pruning, with decrease in number of nodes, the tree overcomes overfitting the training data due to which the accuracies on train and validation data are increases.

(c) Train accuracy: 94.35%
Test accuracy: 77.72%
Valid accuracy: 77.73%



From the graph, we observe that the training accuracy for this part increases more and more with the number of nodes. Also the number of nodes is 10 times more than that in part a because now the numerical attributes are being split on a dynamic median on every level. Thus the numerical attributes are being considered for splitting multiple times.
We observe that around 1800-3000 nodes, we get better accuracies for both the validation and test data than we get in previous parts. But after a certain increase in the number of nodes, the tree starts to overfit the training data and the accuracies for validation and test data decrease significantly. Thus we see that though this serves as a good measure than pre-processing values of numerical attributes in part a, if this is not pruned, it later results in more overfitting than we observed in part a. This is because since we have continuous values of these numeric attributes, as the data gets split more and more, the number of values to decide median on decrease and hence the splitting of tree in later parts become more and more specific to the training data. The numerical attributes 1,3,5,11,12,13 are split multiple times, with 3 being split the most number of times because in the train data, it has most number of distinct values. On 193882 as median, 3 has been split 5 times.

(d) On varying the values of parameters *max_depth*, *min_samples_split*, *min_samples_leaf* from 2-10, 2-10 and 1-15 respectively, I get maximum parameters as:
Max_depth: 12
Min_samples_split: 7
Min_samples_leaf: 4
Train accuracy: 84.56%
Test accuracy: 82.35%
Validation accuracy: 82.33%

We observe that the test accuracy are almost same for our pruned decision tree and scikit-learn library, while the training accuracies are also comparable. The validation

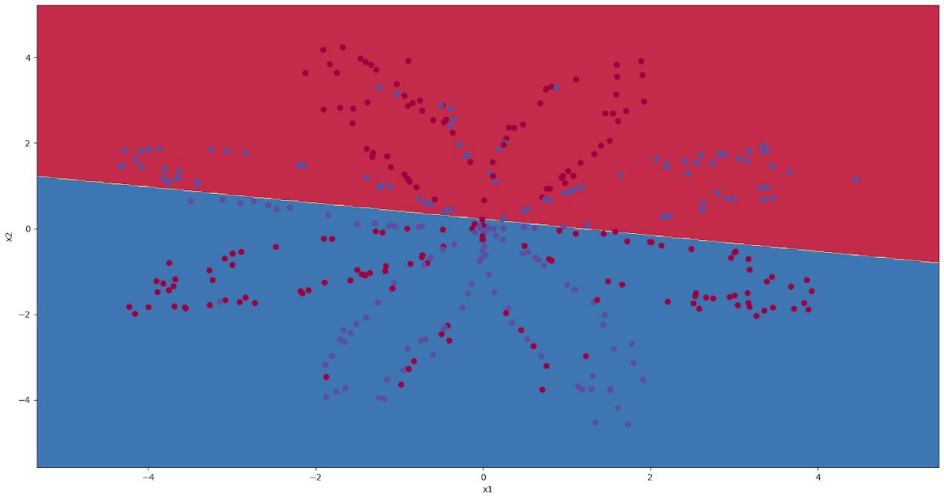
accuracy for our pruned decision tree is better because using first principles, pruning continuously checks accuracy on this validation set and works to optimise that.
For *Max_depth*: 12 and *Min_samples_split*: 7, I observe that the values of the 3 accuracies are almost similar for different values of *min_samples_leaf.* Also a common pattern that is clear among all the variation is that as for fixed values of *max_depth* and *min_samples_split*, as depth increases, accuracy first increase and then after a certain depth, it decreases. For fixed values of *max_depth* and *min_samples_leaf,* as *min_samples_split* increases, accuracy increases and then it saturates.

(e) On varying the values of parameters *n_estimators*, *max_features*, *bootstrap* from 1-15, 1-11 and True/False respectively, I get maximum parameters as:
n_estimators: 13
max_features: 7
bootstrap: True
Train accuracy: 88.66%
Test accuracy: 81.62%
Validation accuracy: 81.46%

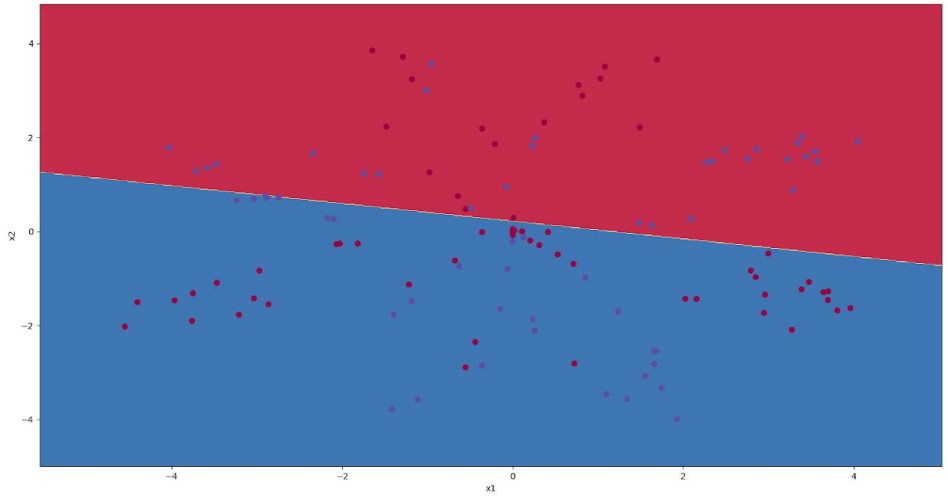
We observe that the performance of our model in part b is better than this model, but this model is better than our model in part c. Random forests model has less variance and hence more stability towards noisy data. This model restricts it's classification to 7 max features to avoid splitting on less important features. The best performance is achieved when there are 13 trees in the forst. The strategy of splitting on every node performs worst among these three because it leads to extreme overfitting as the number of nodes increase a lot.

2) Neural Networks

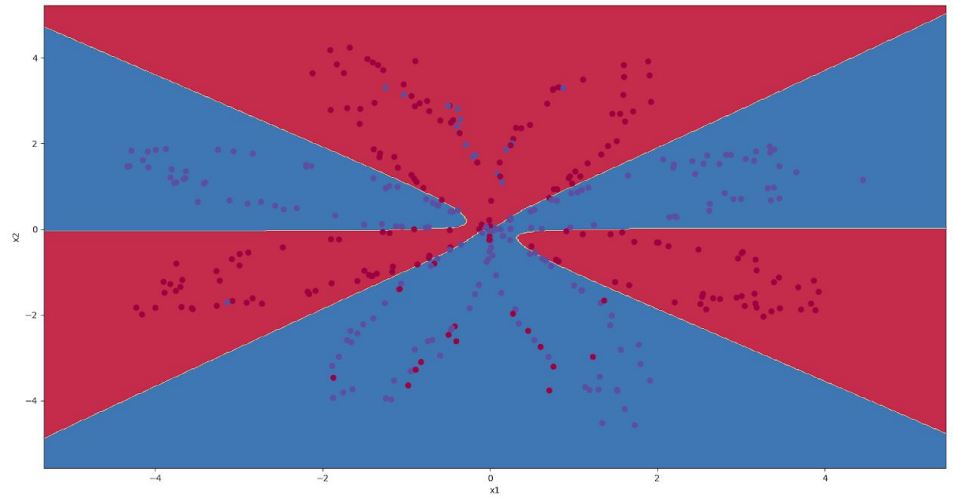
(b) (i) Train accuracy: 45.79%



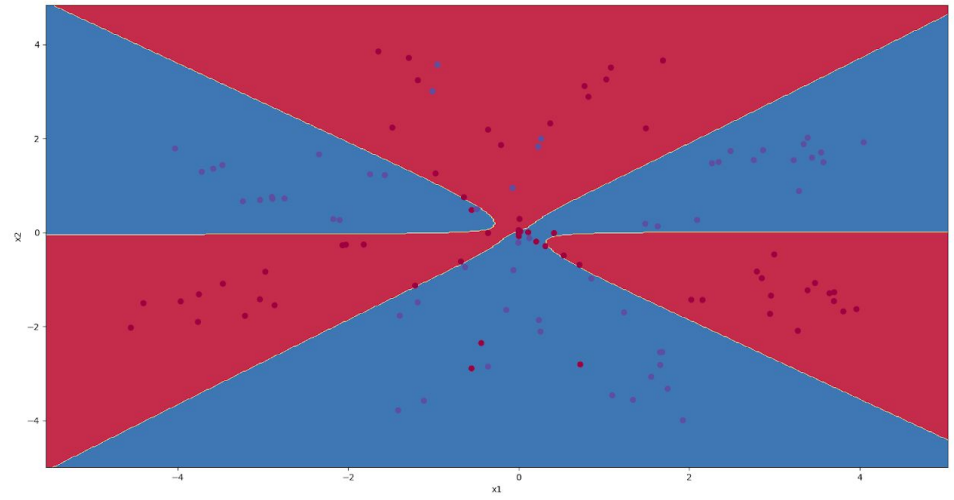
Test accuracy: 38.33%



(ii) Train accuracy: 89.21%



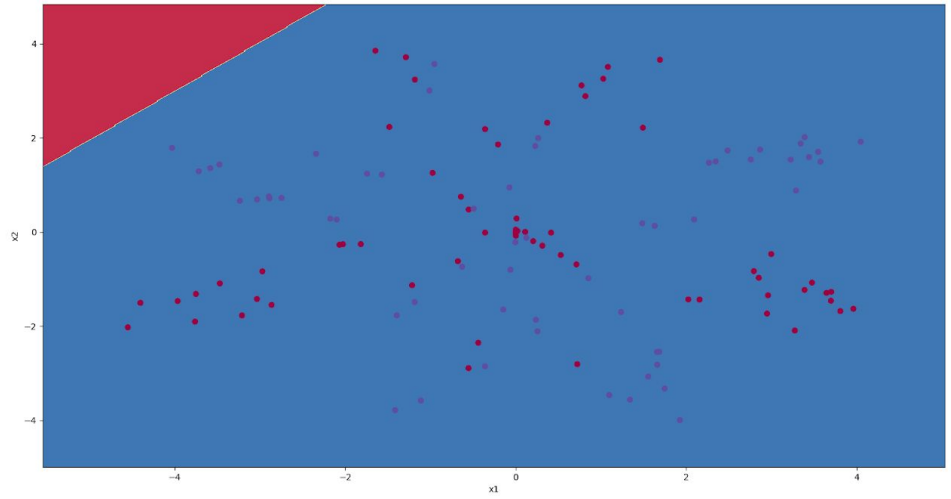
Test accuracy: 85.83%



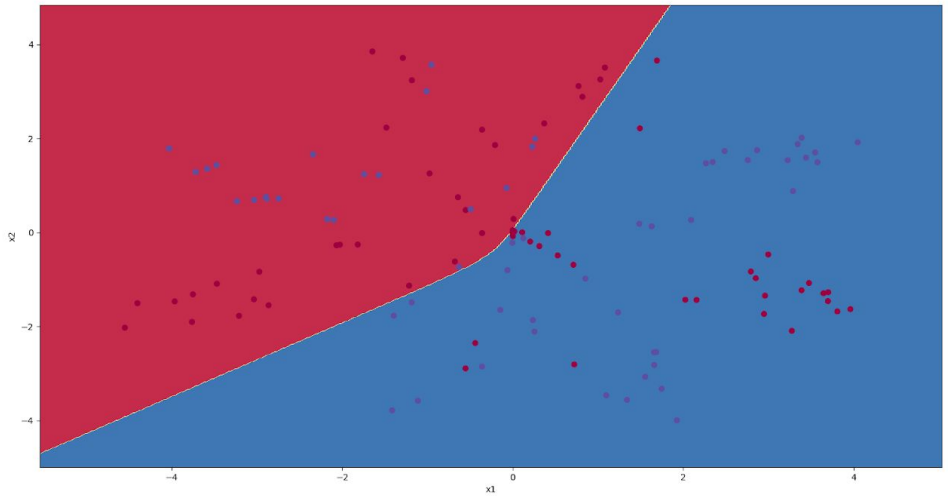
We see that logistic regression classifies the data into two classes (blue and red) using a linear separator, while using neural networks, the decision boundary we obtain is complicated which is clearly not linearly separable. Because of this reason that a log of blue and red points have been wrongly classified giving an accuracy of only 45.79% for train data, which is less than even random prediction. Hence undoubtedly, test accuracy is even less. On the other hand, for decision boundary obtained from neural network we can see that a lot of points have been correctly classified, resulting in such a good training as well as test accuracy (more than double of that obtained by logistic regression).

(iii) Number of units in hidden layer = 1

Train accuracy: 50.52%
Test accuracy: 48.33%

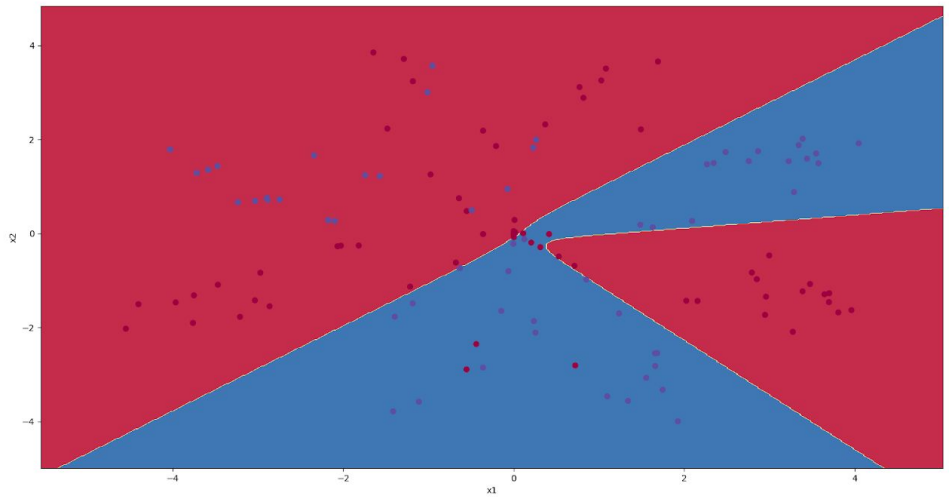


Number of units in hidden layer = 2
Train accuracy: 59.21%
Test accuracy: 57.5%

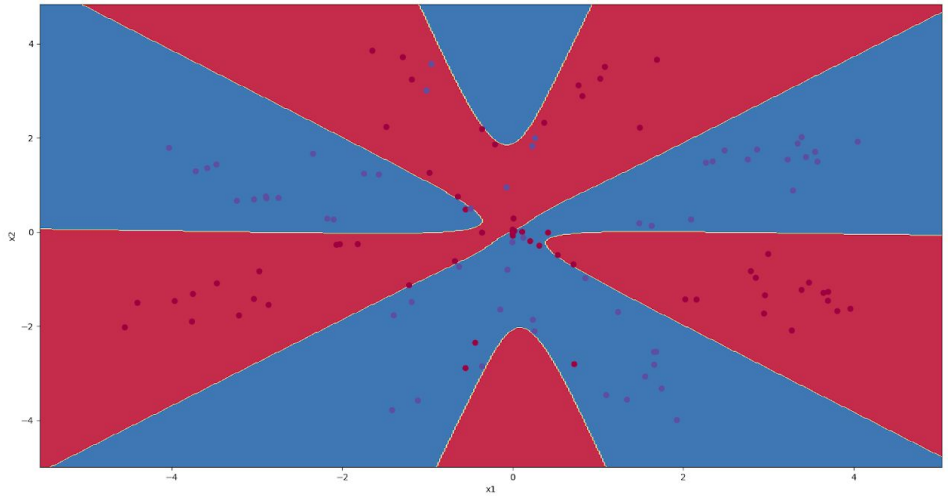


Number of units in hidden layer = 3
Train accuracy: 75.26%

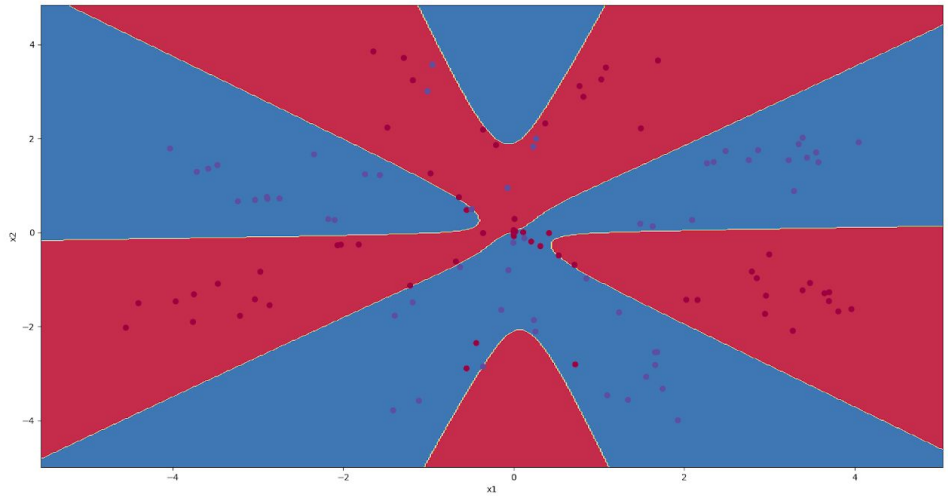
Test accuracy: 77.5%



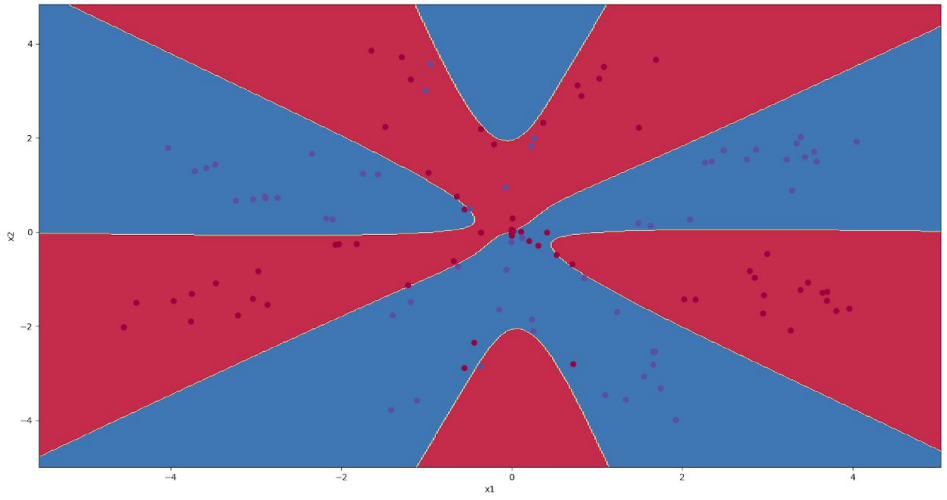
Number of units in hidden layer = 10
Train accuracy: 89.73%
Test accuracy: 85.833%



Number of units in hidden layer = 20
Train accuracy: 90%
Test accuracy: 85%

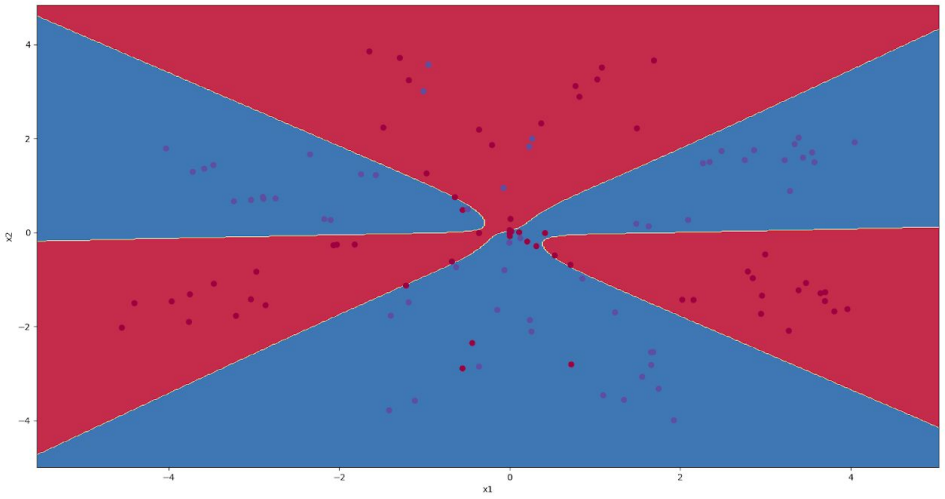


Number of units in hidden layer = 40
Train accuracy: 88.68%
Test accuracy: 83.33%



Initially when the number of hidden units is 1, the model is highly underfit and behaves more like a linear classifier leading to very less accuracies. As we increase the number of units in hidden layer, our decision boundary becomes more and more complicated and hence both test and training accuracies increase. The accuracy increases upto 10 units, but after that the accuracy starts decreasing. This is because the model starts to overfit the training data and find patterns in that are not meaningful.
Thus, the optimal number of units for this problem is 10 because we get the maximum test accuracy of 85.833% in that scenario.

(iv) Number of hidden layers = 2
Train accuracy: 88.68%
Test accuracy: 85%



We see that the accuracies and the decision boundary does not change much even on using 2 hidden layers. This is because accuracy is not proportional to number of hidden layers. Increasing the hidden layers just acts as an overkill.

(c) (i) LIBSVM-
Train accuracy: 99.87%
Test accuracy: 98.8%

NEURAL NETWORK-
Train accuracy: 97.18%
Test accuracy: 96.68%

We find that svm gives better accuracy than neural networks, but the accuracies by both the models are comparable. Since the data is linearly separable, we see that using only 1 perceptron, we can easily represent linear functions.

(ii) Train accuracy: 96.24%
Test accuracy: 95.77%
Stopping Criteria:
To implement SGD, the entire dataset is randomly shuffled. Since the batch size is 100, I use the remaining data as validation set. After each iteration of SGD, new weights are calculated using back propagation. After every update of weights, the algorithm checks accuracy of the model on the validation set. If the accuracy starts increasing, it means that the overall error starts decreasing. After a certain number of iterations for which the accuracy increases on validation set, the SGD iterations stop.
We see the accuracies in this part are comparable but less than in previous part (i). This shows that adding an extra hidden layer does not contribute to the accuracy. Though using learning rate inversely proportional to the square root of number of iterations and using accuracy of validation set for stopping criteria helps escape local minima.

(iii) Train accuracy: 94.71%

Test accuracy: 94.08%

We see that using relu as activation unit behaves as good as sigmoid because ReLu behaves close to a linear unit and does not saturate as per it's definition. But one of the reasons of the accuracy being not more than sigmoid is because units using ReLu as activation function because if activation of any runit become zero after passing through ReLu, then its gradients is associated to zero in back-propagation.