CS5691: Pattern Recognition and Machine Learning
Programming Assignment III

NEURAL NETWORKS & SVM

September 23, 2022

Aanand Krishnan *(BE17B001)*
Manoranjan J *(NA17B112)*
Reneeth Krishna MG *(BS17B025)*
`Team 17 - CCS Section Spring 2021`

Contents

List of Tables

List of Figures

Discriminative Models

Discriminative Models denote a set of models that have a different paradigm of learning from the models we have seen in previous assignments such as GMM's, commonly called *generative models*. Discriminative models focuses on the task at hand, that is to discriminate between classes and does so by directly trying to learn the boundary surface between them. Unlike generative models which model the distribution of individual classes, it doesn't assume any underlying distribution. In short, discriminative algorithms allow us to classify points, without providing a model of how the points are actually generated.

Mathematical Structure Comparisons

For illustration purposes, consider a classification model with a training data-set of N-samples, $X = \{\overline{x_1}, \overline{x_2}, ..., \overline{x_N}\}$ with known labels for $K-$classes denoted as, $Y = \{1, 2, ..., K\}$. Our goal after training is to estimate the posterior probabilities, $P(Y|X)$ to make a prediction on an unknown data point.

To estimate the posterior probability, *generative models* make use of the Bayes rule using the prior probability $P(Y)$ of the classes and likelihood $P(X|Y)$ estimated using the training data-set. In simpler terms we have to make an estimate for the joint probability distribution $P(X, Y)$ which is transformed to get $P(Y|X)$. The Bayes rule is given by,

$$P(Y|X) = \frac{P(X|Y).P(Y)}{P(X)}$$

Which can further be written as,

$$P(Y = y|X) = P(y).P(\overline{x_1}|y).P(\overline{x_2}|y, x_1)...P(\overline{x_N}|y, x_1, x_2, ..., x_{N-1})$$

To simplify the complexity of the above equation we would have to make a further justified assumption that all $\overline{x_i's}$ are conditionally independent. The posterior probability then takes the familiar form,

$$P(Y = y|X) = \frac{P(y).P(\overline{x_1}|y).P(\overline{x_2}|y)...P(\overline{x_N}|y)}{P(X)}$$

In contrast to this, *discriminative models* infer $P(Y|X)$ directly from the given set of training examples. Discriminative models in this sense are straigh forward and don't require solving an intermediate problem as seen for generative models. An intuitive way of thinking about this kind of models is finding the decision boundaries that divide the space in to various classes. This behavior is very clear in neural networks a type of discriminative modle, where the computed weights can be seen as a complexly shaped curve isolating the elements of a class in the space. Support Vector Machines (SVM's) are a more sophisticated descriminative model which identifies the optimal boundary curves.

Neural Networks

Neural networks are biologically-inspired method for building computer programs that has an entirely different framework compared to conventional programs. They can learn and independently find connections in data. They are much simpler versions of the biological neurons. Each neuron in a neural net is characterized by a set of synapses or connecting links which has certain weights associated with it. All the input signals are added up with the *bias* term after being properly weighed and there is also a function called the *activation function* denoted as $\phi(a)$ that serves as output of a neuron by taking in the weighed inputs. The outputs produced by the activation function are generally normalized and lies between either $[0, 1]$ or $[-1, 1]$. A distinguishing factor among various neurons are the number of connecting links and hidden layers, learning process along with the activation function used.

Support Vector Machines

Support Vector Machines are the most commonly used efficient method for classification task. It finds the optimum hyper-plane, that is the global maximum unlike neural networks. Fore illustration purposes consider an training dataset of two linearly separable classes, $D = \{(x_n, t_n)\}_{n=1}^{N}$. An optimal separating hyper-plane between two classes maximises the distance of the nearest examples of either classes. Distance to the nearest training example is called the margin of the separating hyper-plane. For a unique solution we define a canonical hyper-plane which satisfies the following condition,

$$|g(\overline{x}^*)| = 1$$

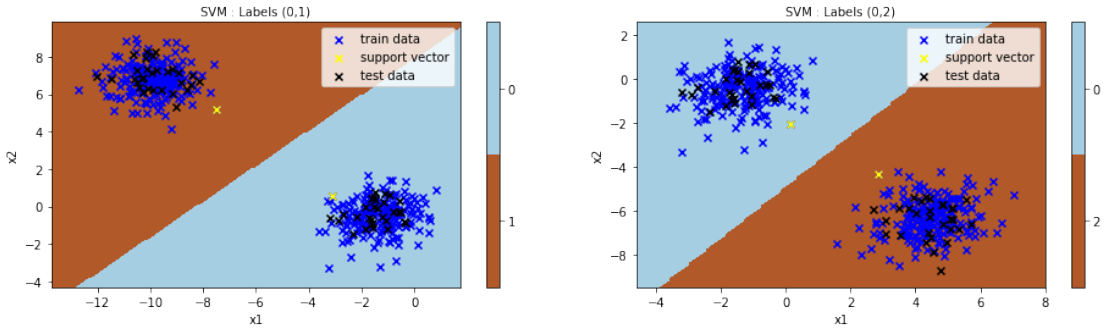
The constraint for Optimization is accounted by using Lagrange multiplier method. Following which, we have to solve a primal-dual problem to obtain the optimal solution. A salient feature to be noted here is that learning and classification depends only on dot products of sample pairs and dot products of unknown with the sample. Such a reliance on dot products enables approach to problems in which samples cannot be separated by a straight line. This method involves transformation of such classes in to a higher dimension space where they become linearly separable. In doing so, because of the reliance only on dot products, we need not know the transformation themselves. Knowing the functions called kernels which give the dot product is sufficient. Some standard choices of kernel are Gaussian, polynomial and radial.

I. Pattern classification on Linearly Separable Data

Dataset 1A provided consisted of 2D data and all the data points were found in well organized clusters corresponding to each label and were observed to be linearly separable on plotting.

I.1. Linear SVM for every pair of classes

SVMs with only the linear kernel implementation were used from the scikit library to discriminate between two given class labels which were trained on all 4C2 combinations of the class labels, with thorough functionality being achieved as the data points were already linearly separable in their default linear feature space.



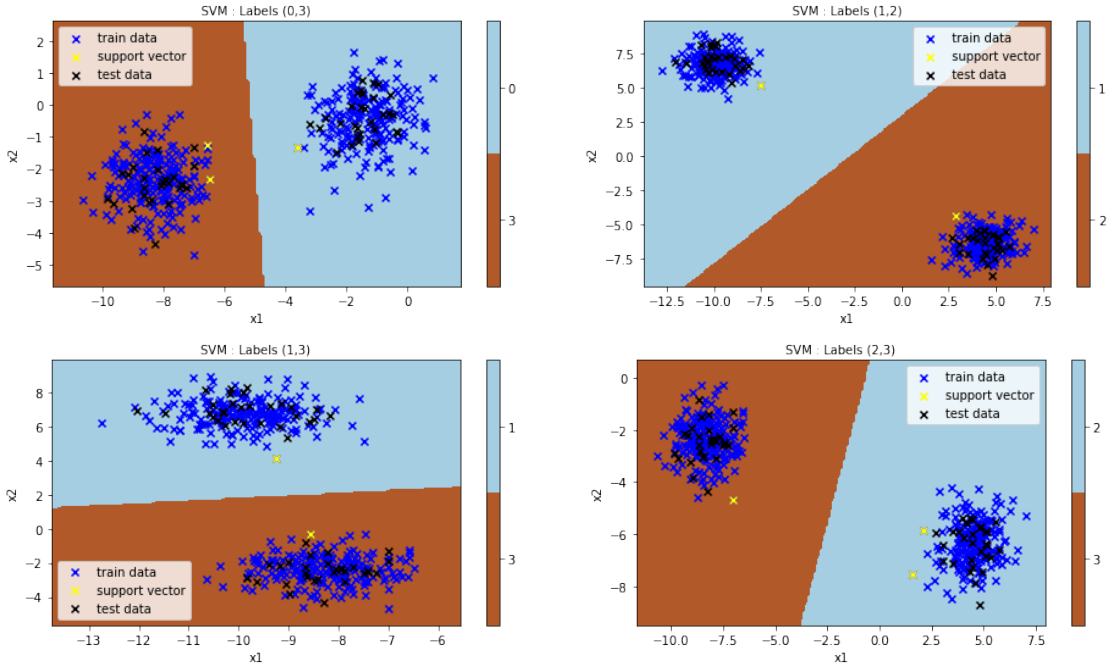


Figure 2: Various Decision Plots for svm classifier involving two classes

I.2. Perceptron for every pair of Classes

A single layer perceptron to discriminate between two given class labels were trained on all ${}^4C_2$ combinations of the class labels. With only a single node in the output layer the class labels 't' consists of $+1/-1$ for each class.

The perceptrons with only the implementation of linear basis function(including addition of bias term)in all cases were found to converge to 100% training and validation accuracy within a single epoch and a learning parameter of 0.1.This can be observed in the plots by establishment of clear linear separation boundaries in the contour plots with the superposed data.In the case of labels(0,2), the inability of the perceptron to form a clear margin separation was observed due to the proximity of the data points.

It was observed with the use of lower learning rates eta was associated with poorer separation between the data points and boundaries,where the thinning of the margin between the separation layer and data points can be observed which resulted in lower classification accuracies.

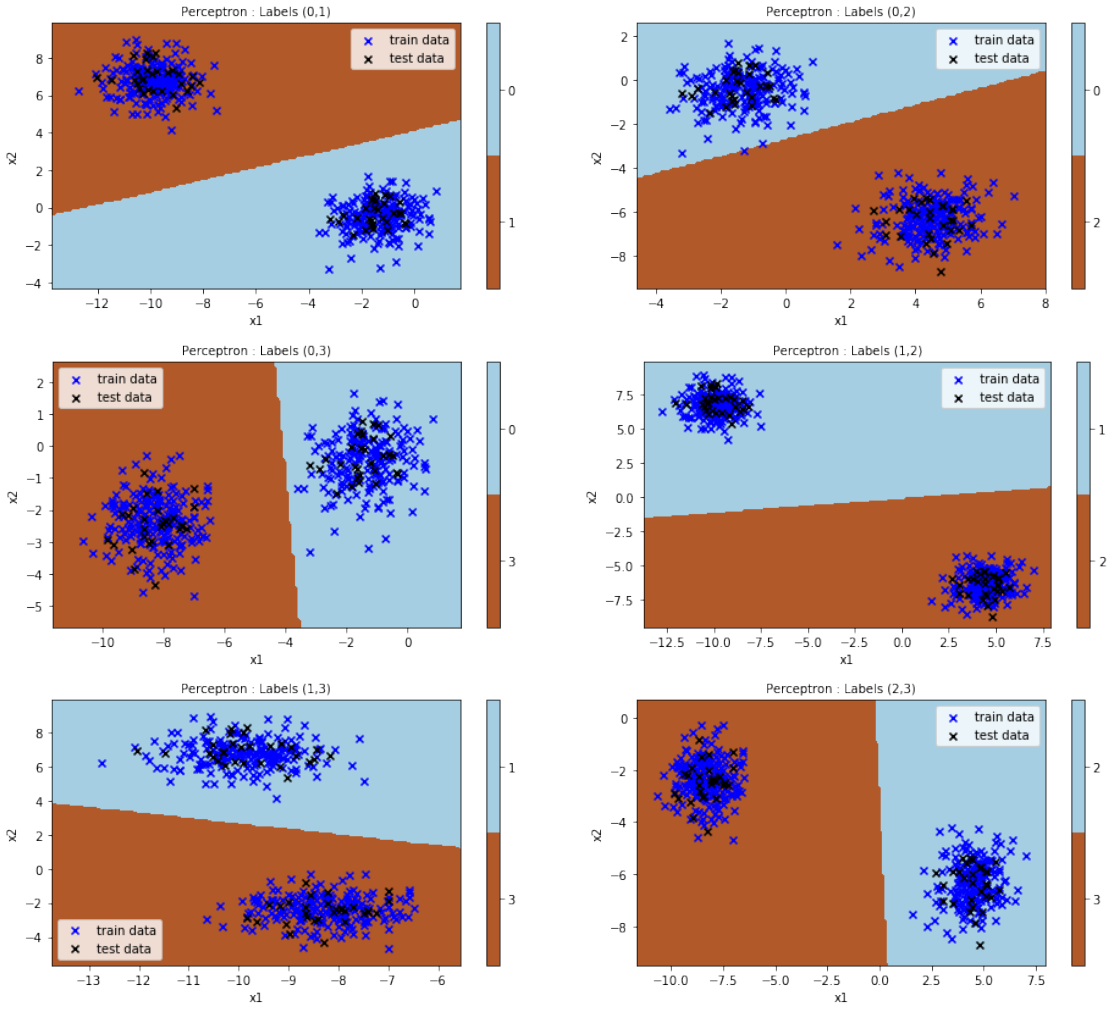
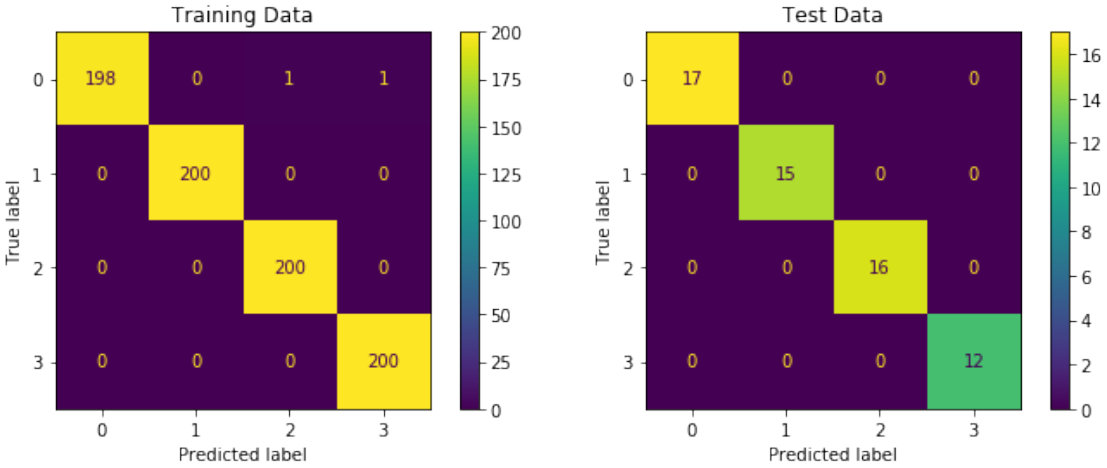


Figure 3: Various Decision Plots for perceptron classifier involving two classes

I.3. MLFFNN with a single hidden layer for all classes

MLFFNN with a single hidden layer was used to classify all four labels of the data simultaneously unlike the perceptron by using $1 - K(k = 4)$ one hot encoding for target y_i. The MLFFNN model was implemented with the pytorch library. The operation of the model consisted of using the CSE(Cross Entropy) error function and weight updation through stochastic gradient descent achieved through the back propagation algorithm. Different models corresponding to different types of activation functions, number of nodes in the hidden layer and with loss function convergence

tolerance of $1e-5$ were tested on the data and the performance was tabulated in Table[3].



(a) Confusion Matrix for training data (b) Confusion Matrix for test data

Figure 4: Confusion Matrix for the best model

In all cases ,the usage of sigmoid activation function was found to have the lowest classification accuracy overall. Significant improvement in classification accuracy in all cases was observed with the addition of the fourth node in the hidden layer.The Soft-max and ReLU activation function were noted for its ability to form separation boundaries which were nearly of linear shape in the decision region plots.The best model with 100 % validation accuracy and the highest training of 99.875 was found to be the MLFNN with four hidden nodes utilizing the ELU activation function.

Hidden Layer Nodes = 2		
Activation Function	Training Accuracy (%)	Test Accuracy (%)
Sigmoid	68	70
Hyperbolic Tangent	95.75	99.167
ReLu	91.625	91.67
ELU	99.5	100

Hidden Layer Nodes = 3		
Activation Function	Training Accuracy (%)	Test Accuracy (%)
Sigmoid	72.875	75
Hyperbolic Tangent	98.875	100
ReLu	99.625	100
ELU	99.875	100
Hidden Layer Nodes = 4		
Activation Function	Training Accuracy (%)	Test Accuracy (%)
Sigmoid	99.875	100
Hyperbolic Tangent	99.375	100
ReLu	99.875	100
ELU	99.875	100

Table 1: Performance of Various MLFFNN models

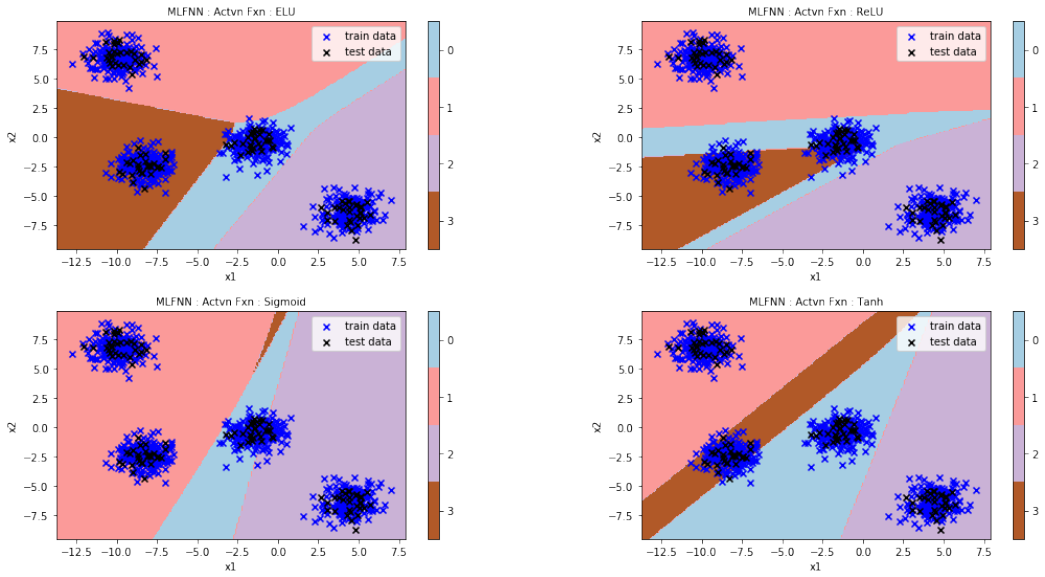


Figure 5: Various Decision Plots for mlfnn classifier with hidden layer nodes : 2

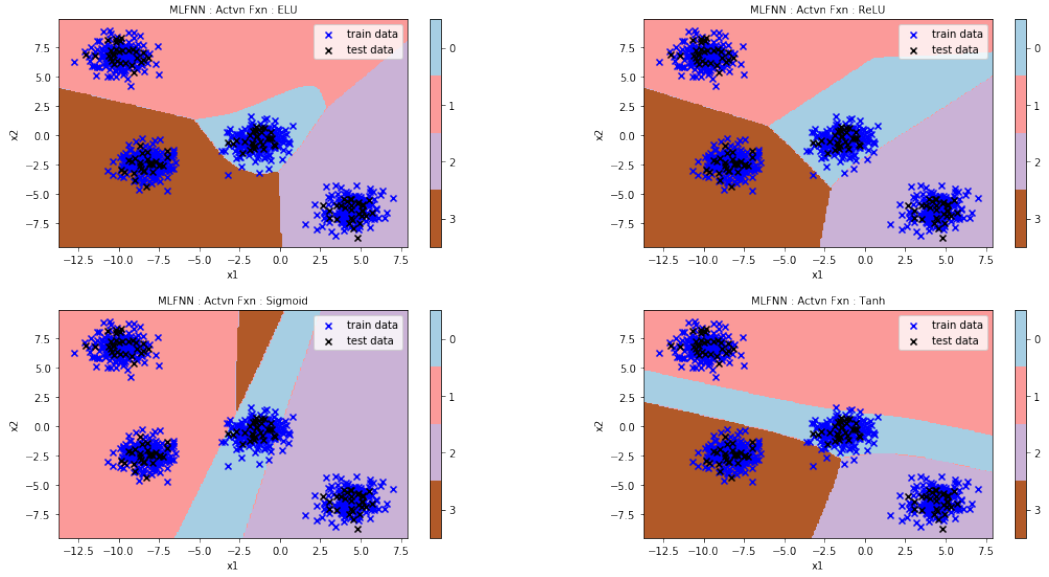


Figure 6: Various Decision Plots for mlfnn classifier with hidden layer nodes : 3

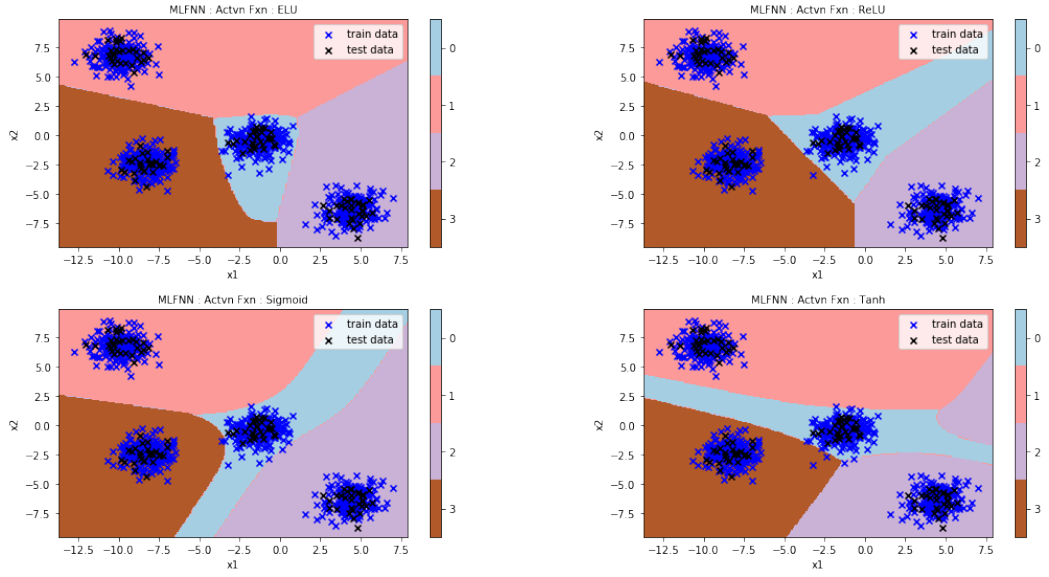


Figure 7: Various Decision Plots for mlfnn classifier with hidden layer nodes : 4

II. Pattern classification on Non-linearly Separable Data

II.1. MLFFNN with 2 hidden layers

The given synthetic data is classified using Multi Layer Feed Forward Networks with 2 hidden layers. The Neural Network is built using pytorch. We experimented with the number of nodes in each hidden layer. Cross Entropy Loss is used for all the models built since it is known to perform better for conventional classification problems. ReLu Activation layer is used for the hidden layers. Let us look at the difference performance characteristics of the models. A learning rate of 0.001 is kept throughout. Here [x y] represents the number of hidden layer nodes in the 2 hidden layer used.

Model	Training Accuracy	Val Accuracy
[2 2]	73.68%	80.28%
[2 3]	77.72%	87.5%
[3 3]	98.02%	100%
[4 4]	98.67%	100%

Table 2: Performance of various MLFFNN Models

.

As we increase the number of neurons, we can observe that the accuracy increases. Another big factor is the learning rate. If we have a larger learning rate, then there are lots of oscillations when the solution is converging. If it is too small the solution will take time to converge. For choosing a good model, we chose a model that doesn't oscillate much during convergence and thus we kept a learning rate of 0.001. Since learning rate is low, the number of epochs for convergence is high. The test accuracy for the best model with 4 hidden layer nodes in each layer was 100%. Another important point to note is that the same neural network model when retested the performance may vary slightly. This is because each time we run it, the weight initialization will be different and thus the optimal value it reaches will also be different. The loss curve, confusion matrix and the decision plot of the best model is given.

Looking at the decision region,we see how neural networks try to capture non-linearities in the data. It performs well in the region wherever data is available for it

to learn.

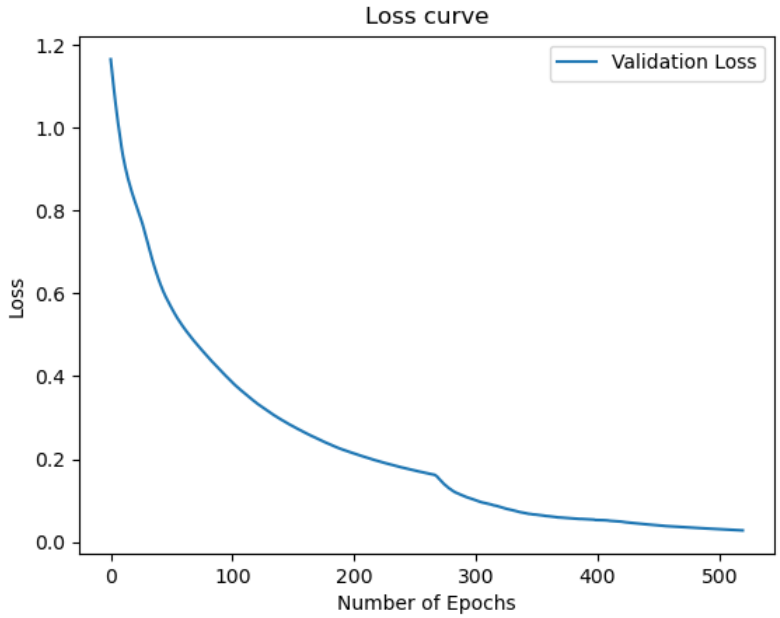
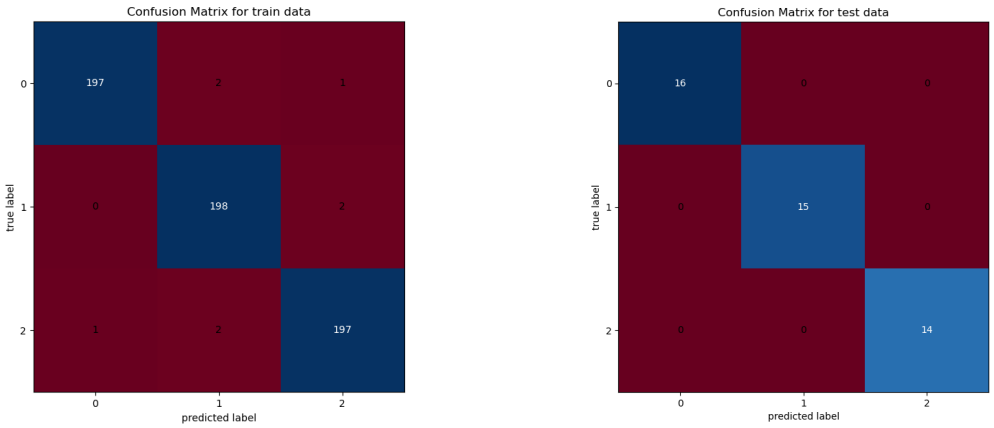


Figure 8: Validation Loss curve



(a) Confusion Matrix for training data (b) Confusion Matrix for test data

Figure 9: Confusion Matrix for the best model

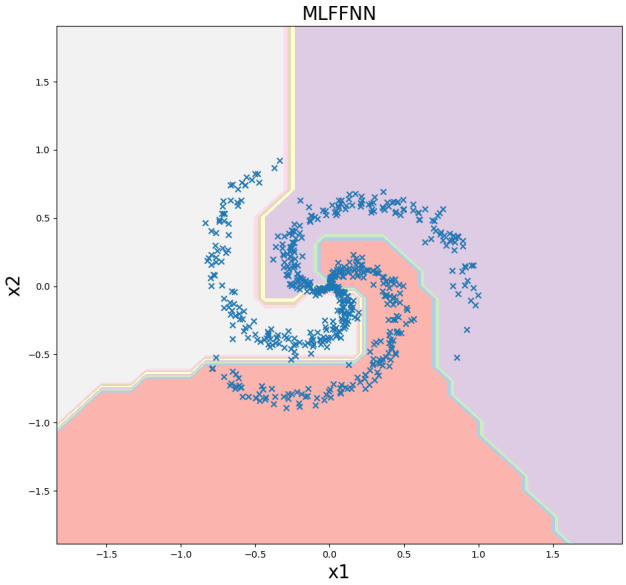


Figure 10: Decision plot for the best model

II.1.1. Surface plots for different nodes

In this section we are going to see surface plots for the nodes of hidden layers and output layers. Totally we have used 8 nodes in the 2 hidden layers and since there are 3 classes we are using 3 nodes in the output layers. So 11 nodes in total. We have used ReLu activation function in the hidden layers and are plotting the values after the activation functions. In the output layer we have not used it, since the cross entropy function of pytorch requires it to be a raw output from the nodes. We plot the outputs after 1,5,20,100 epochs and after convergence (500 epochs).

Now looking at the surface plots, we can get some understanding of how the neural network takes a decision. Based on the output values of the hidden layers, the values in the output layer changes and based on this the decision changes. This ability to decide, that is the probability to tell that a data point belongs to the class strongly increases with number of epochs.

Epoch 1

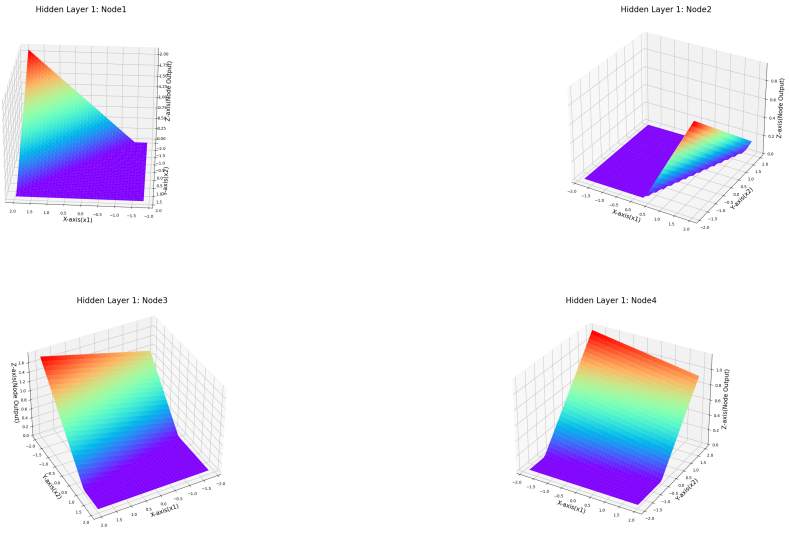


Figure 11: Surface Plots for Hidden Layer 1 - Epoch 1

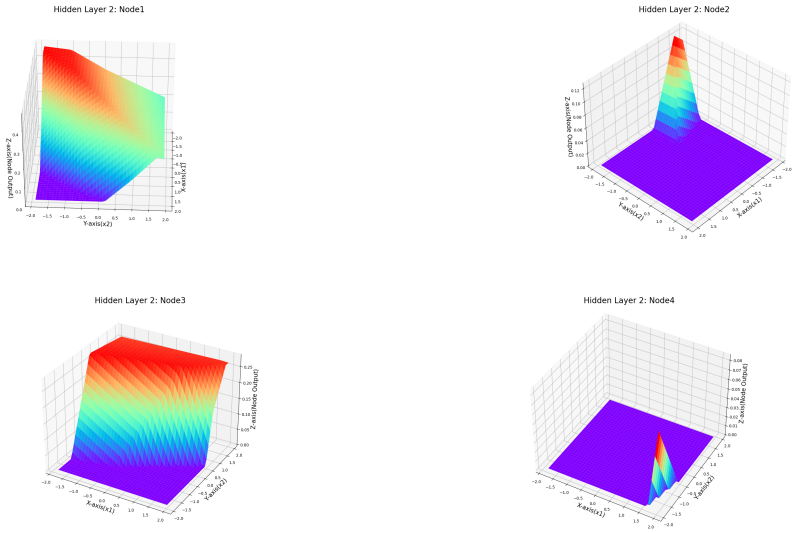


Figure 12: Surface Plots for Hidden Layer 2 - Epoch 1

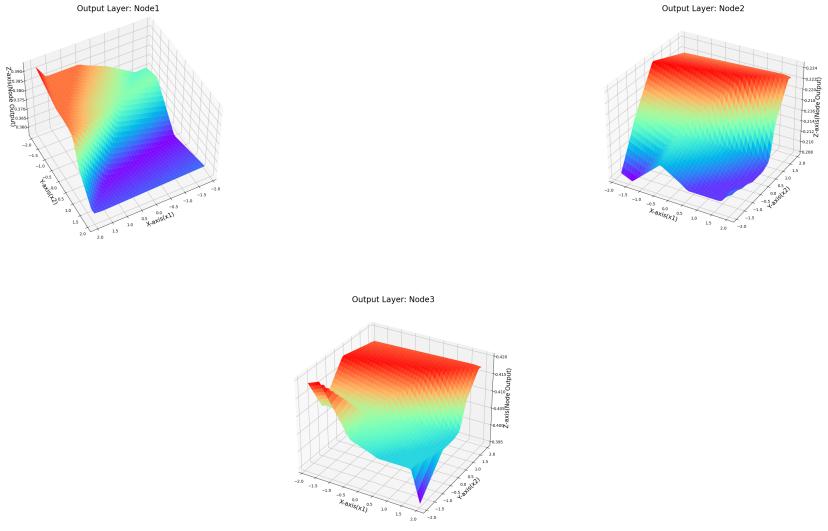


Figure 13: Surface Plots for Output Layer - Epoch 1

Epoch 5

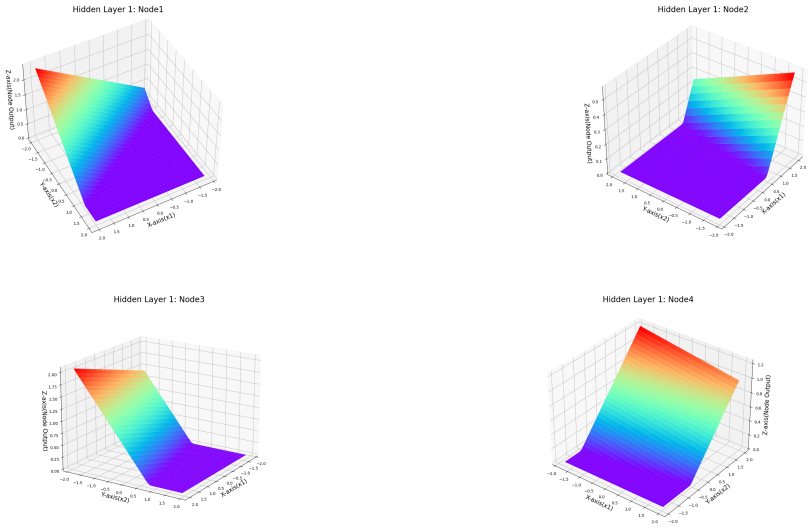


Figure 14: Surface Plots for Hidden Layer 1 - Epoch 5

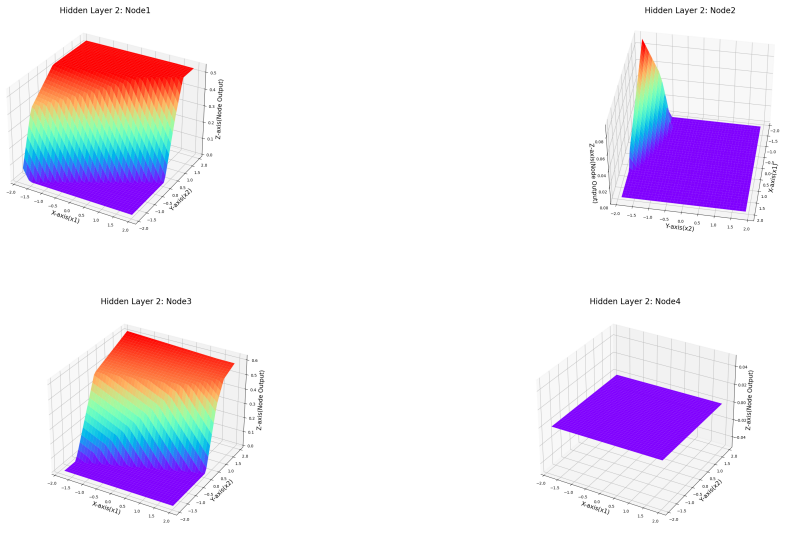


Figure 15: Surface Plots for Hidden Layer 2 - Epoch 5

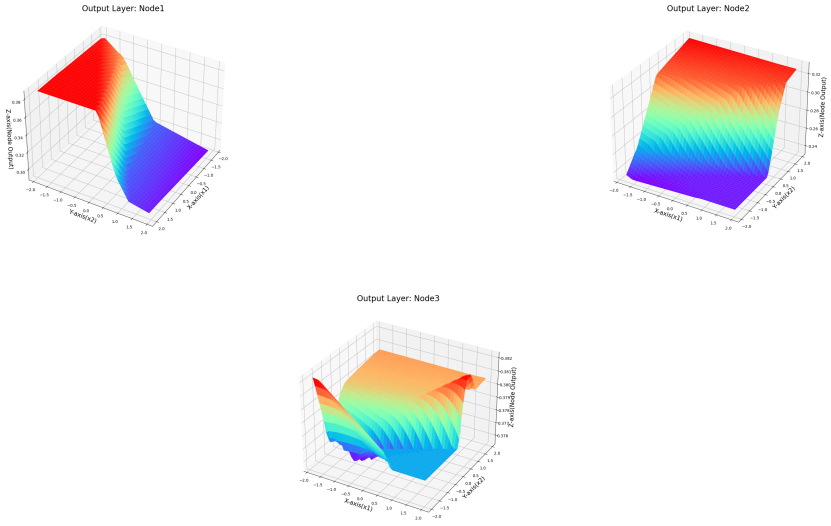


Figure 16: Surface Plots for Output Layer - Epoch 5

Epoch 20

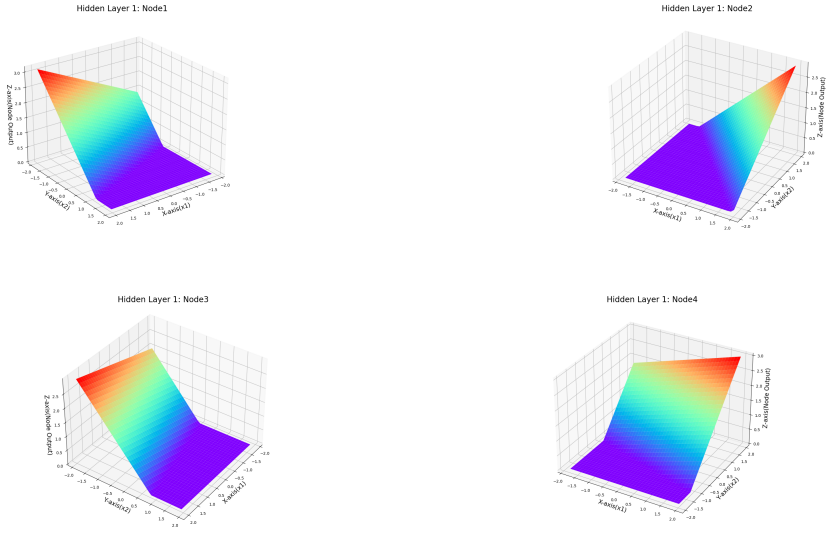


Figure 17: Surface Plots for Hidden Layer 1 - Epoch 20

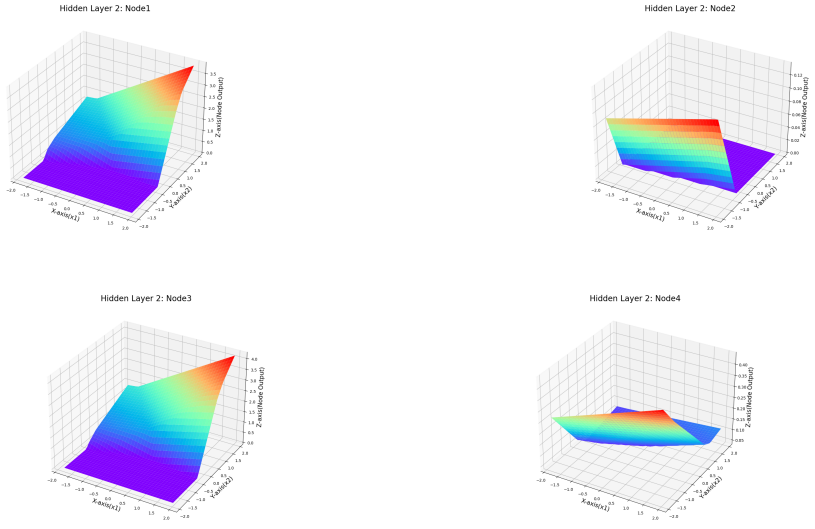


Figure 18: Surface Plots for Hidden Layer 2 - Epoch 20

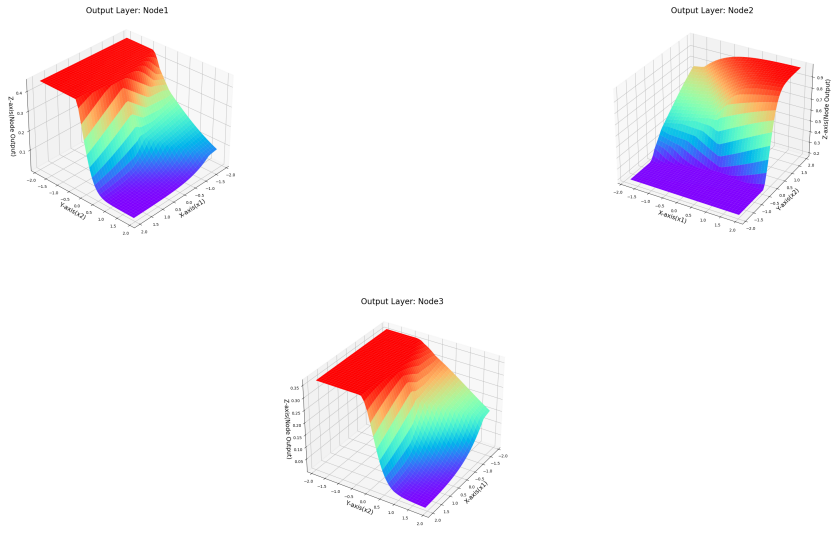


Figure 19: Surface Plots for Output Layer - Epoch 20

Epoch 100

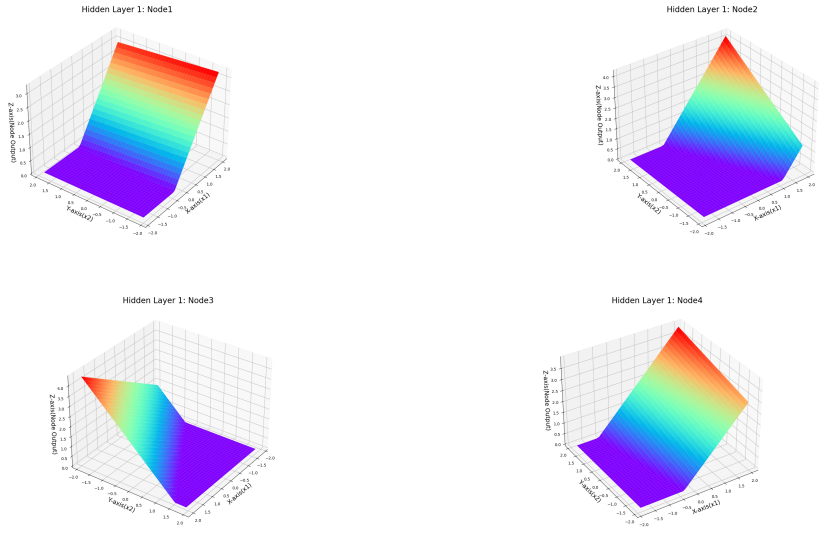


Figure 20: Surface Plots for Hidden Layer 1 - Epoch 100

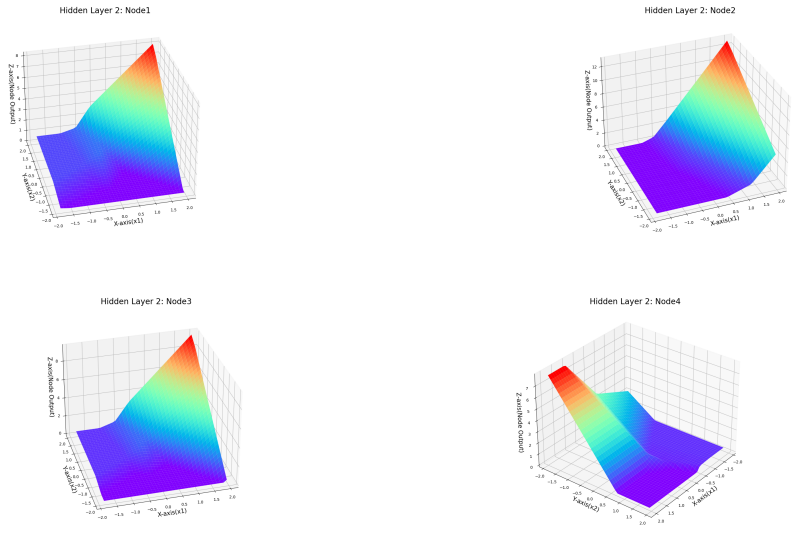


Figure 21: Surface Plots for Hidden Layer 2 - Epoch 100

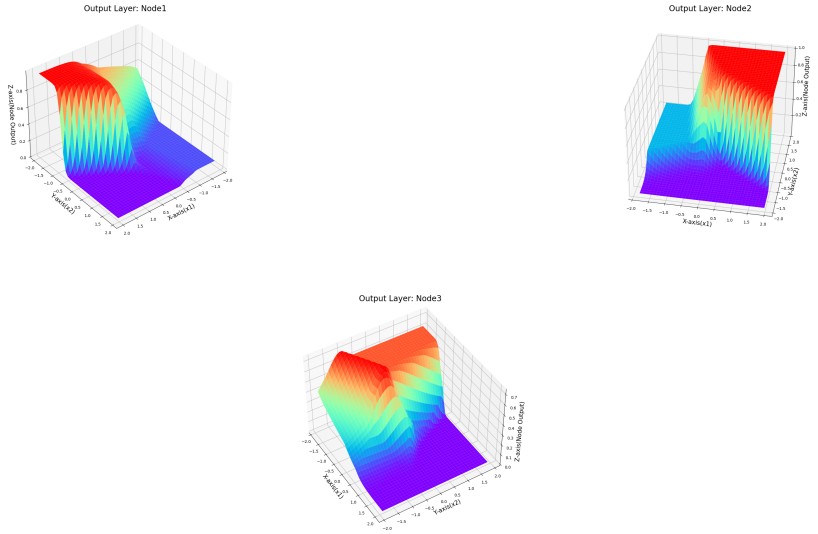


Figure 22: Surface Plots for Output Layer - Epoch 100

Epoch 500

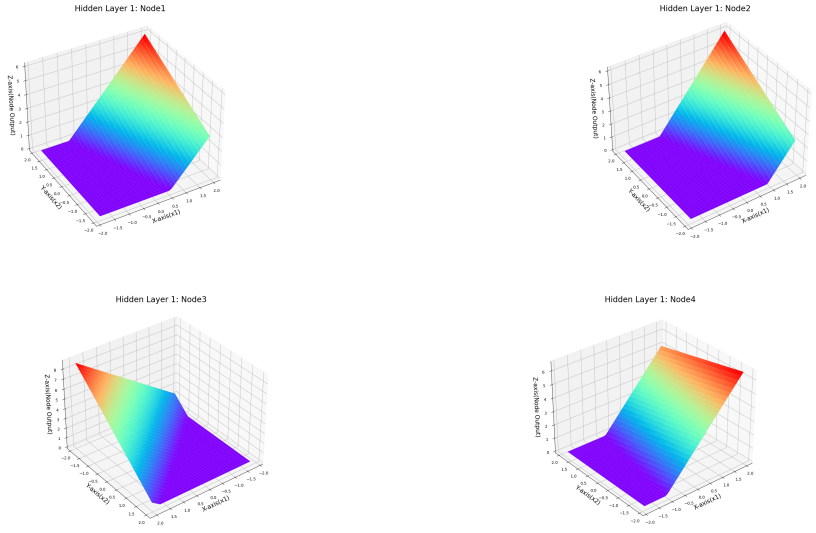


Figure 23: Surface Plots for Hidden Layer 1 - Epoch 500

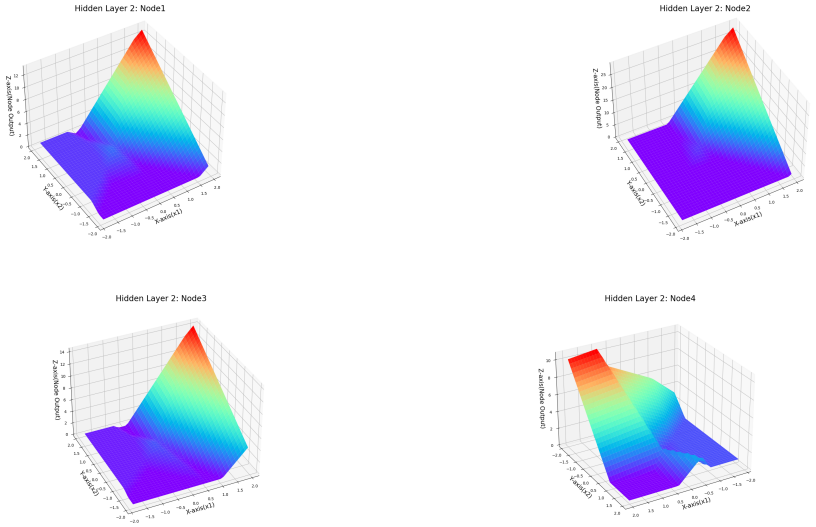


Figure 24: Surface Plots for Hidden Layer 2 - Epoch 500

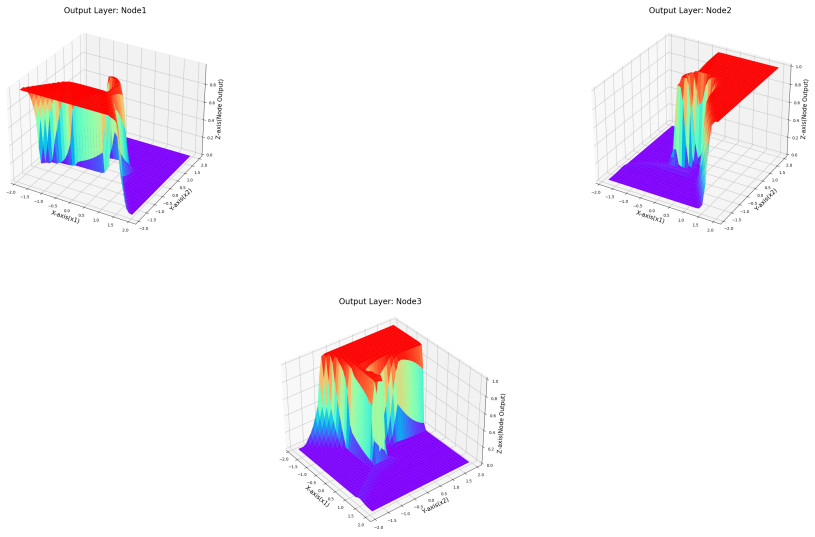


Figure 25: Surface Plots for Output Layer - Epoch 500

II.2. Non-Linear SVM with One-Against-The-Rest Approach

The given synthetic data is classified using Support Vector Classifier with One Vs the Rest Approach. Sklearn was used to implement the method. It was tried with polynomial and gaussian kernels. Let us look at the performance of the various models by changing different parameters.

Both the models can handle the data very well and the best model in both the cases gave a 100% test accuracy. In case of gaussian kernels, gamma as 10 gave the best model where gamma is the width parameter. In the case of polynomial kernels, setting degree as 4, coeff as 1 and gamma as 10 gave the best model. In both the cases, the validation accuracy was also 100%.

Model	Training Accuracy	Val Accuracy
C=1	98.34%	100%
C=0.5	97.84%	100%
C=2	99.17%	100%
C=4	99.5%	100%
gamma=0.1	52.17%	68.89%
gamma=1	90.5%	91.12%
gamma=10	99.67%	100%

Table 3: Performance of various SVM Models with gaussian kernels

.

Model	Training Accuracy	Val Accuracy
Deg=2 / coeff=0	36.17%	31.12%
Deg=2 / coeff=1	63.0%	73.34%
Deg=2 / coeff=1 / gamma=1	57.49%	68.89%
Deg=3 / coeff=1 / gamma=10	99.17%	100%
Deg=4 / coeff=1 / gamma=10	99.67%	100%

Table 4: Performance of various SVM Models with polynomials kernels

.

II.2.1. Plots for Gaussian Kernel SVM

(a) Confusion Matrix for training data (b) Confusion Matrix for test data

Figure 26: Confusion Matrix for the best model

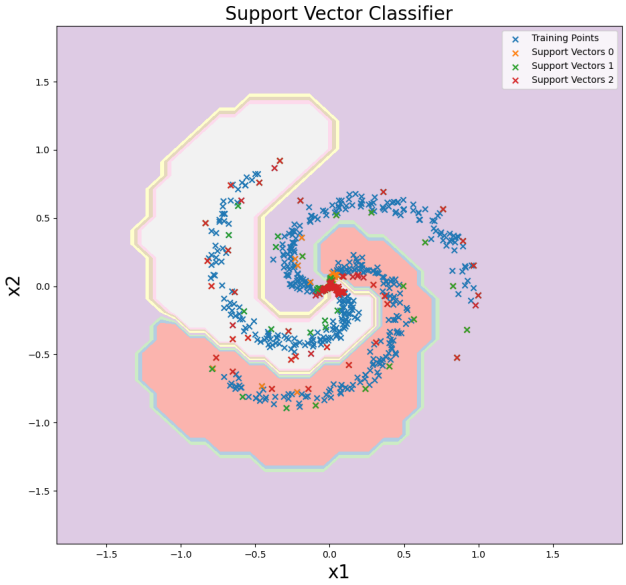


Figure 27: Decision plot for the best model

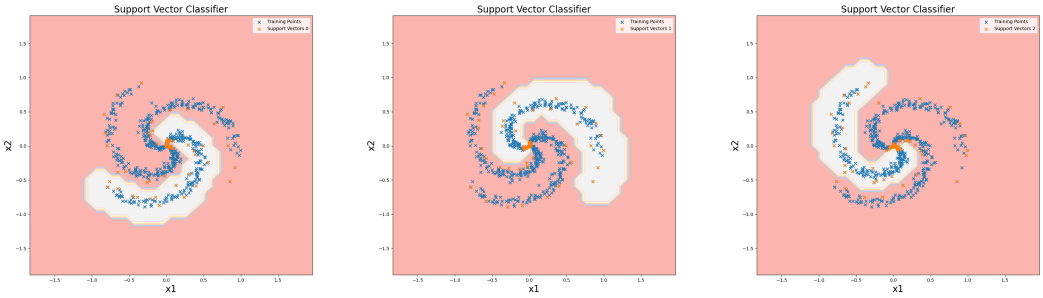
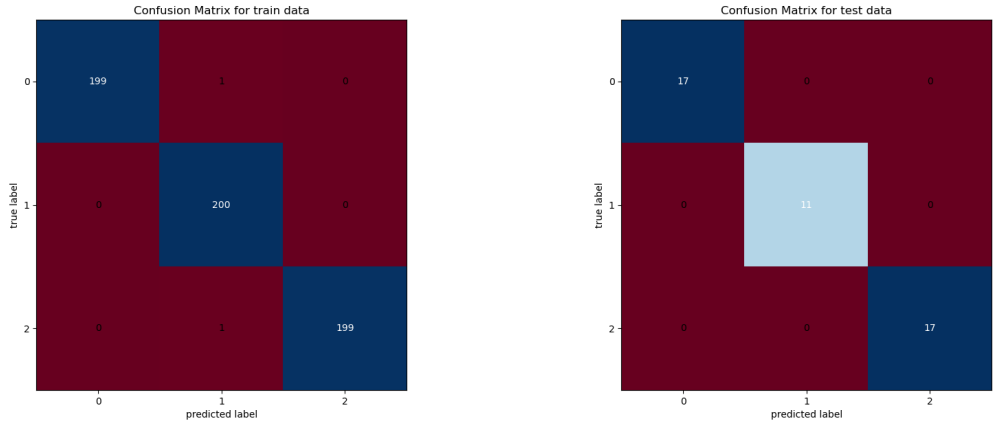


Figure 28: Various Decision Plots for 3 classifiers used in One Vs Rest Approach

II.2.2. Plots for Polynomial Kernel SVM



(a) Confusion Matrix for training data (b) Confusion Matrix for test data

Figure 29: Confusion Matrix for the best model

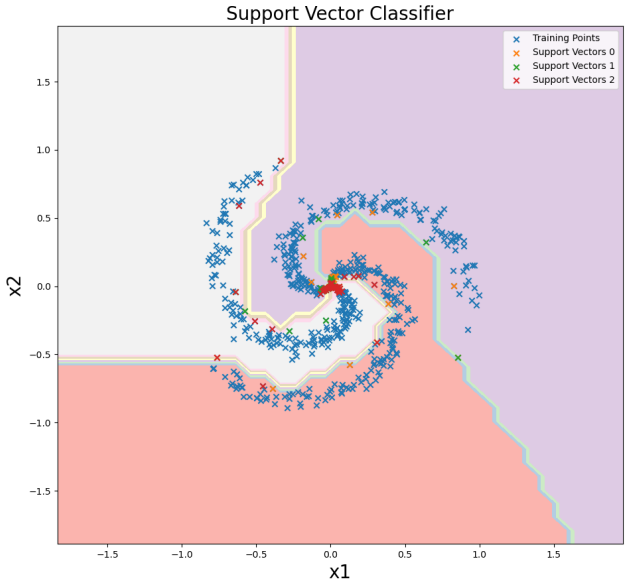


Figure 30: Decision plot for the best model

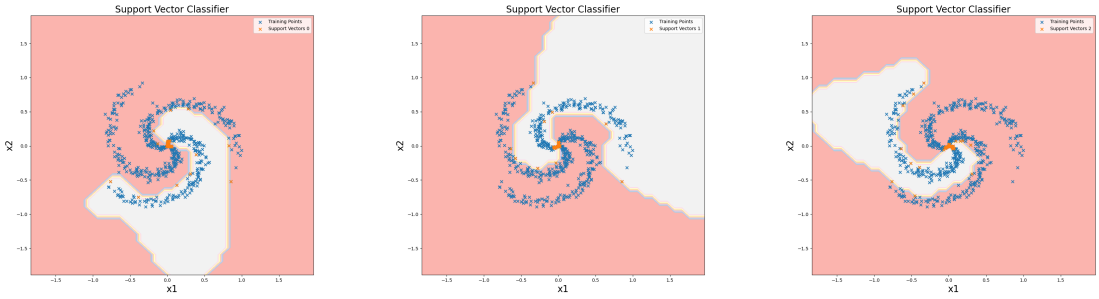


Figure 31: Various Decision Plots for 3 classifiers used in One V sRestApproach

III. Static Pattern classification on Real World Data

III.1. MLFFNN with 2 hidden layers

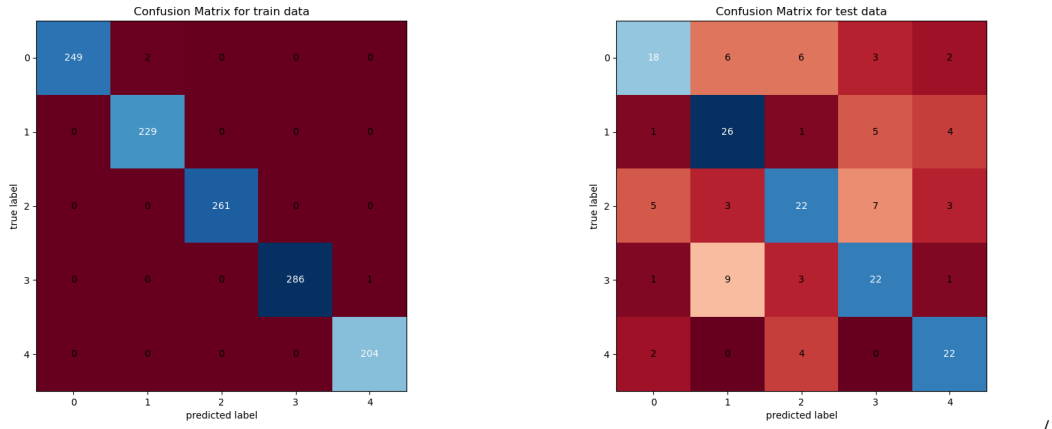
The given real world data is classified using Multi Layer Feed Forward Networks with 2 hidden layers. The Neural Network is built using pytorch. We experimented with the number of nodes in each hidden layer. Cross Entropy Loss is used for all the models built since it is known to perform better for conventional classification problems. ReLu Activation layer is used for the hidden layers. A learning rate of 0.001 is kept constant. Let us look at the difference performance characteristics of the models. [x y] refers to the number of nodes in the hidden layer.

It was observed that as the number of nodes in hidden layer increased, the performance increases. Similarly as the number of epochs increases, the model tends to learn better. On the other hand increasing the learning rate results in poor performance and there will be so much oscillations during convergence. The activation function is kept as ReLu for the hidden layers. After increasing the number of nodes, the model starts to overfit. The test accuracy is 64.66% for the best model.

Model	Training Accuracy	Val Accuracy
[20 40]	64.42%	58.27%
[80 40]	67.06%	60.24%
[120 80]	77.14%	61.71%
[500 250]	99.91%	63.91%

Table 5: Performance of various MLFFNN Models

.



(a) Confusion Matrix for training data

(b) Confusion Matrix for test data

Figure 32: Confusion Matrix for the best model

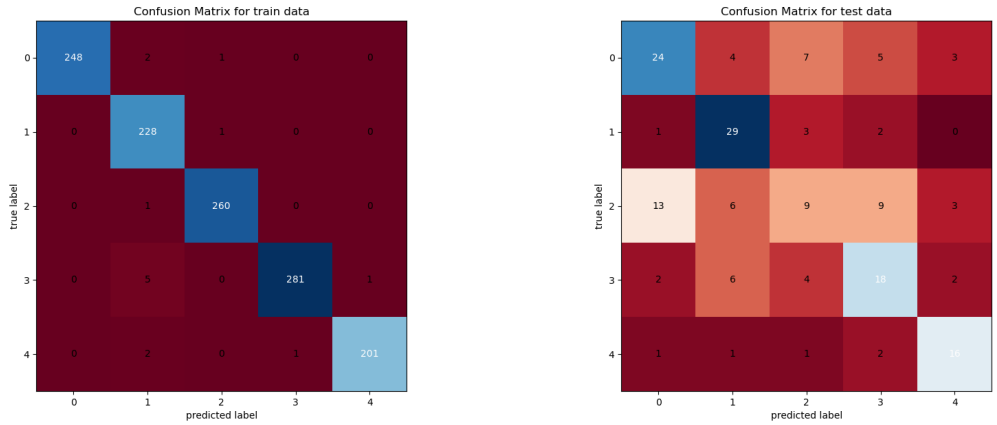
III.2. SVM with Gaussian Kernel

The given real world data is classified using Support Vector Classifier with One Vs the Rest Approach. Sklearn was used to implement the method. It was tried with gaussian kernels. Let us look at the performance of the various models by changing different parameters. The best model was observed for gamma=20 which is the width parameter and the test accuracy was 56.14%.

Model	Training Accuracy	Val Accuracy
C=1	78.24%	55%
C=3	85.47%	57.78%
C=6	90.01%	57.22%
C=10	93.26%	53.33%
C=15	95.29%	56.11%
gamma=1	67.53%	51.92%
gamma=10	93.83%	58.46%
gamma=20	98.86%	62.84%

Table 6: Performance of various SVM Models with gaussian kernels

.



(a) Confusion Matrix for training data (b) Confusion Matrix for test data

Figure 33: Confusion Matrix for the best model