Neural Network Laboratory Work – 2

Ravinthiran Partheepan

Single Layer Perceptron

- A **single layer perceptron** is a feed-forward network based on a Sigmoid transfer function.

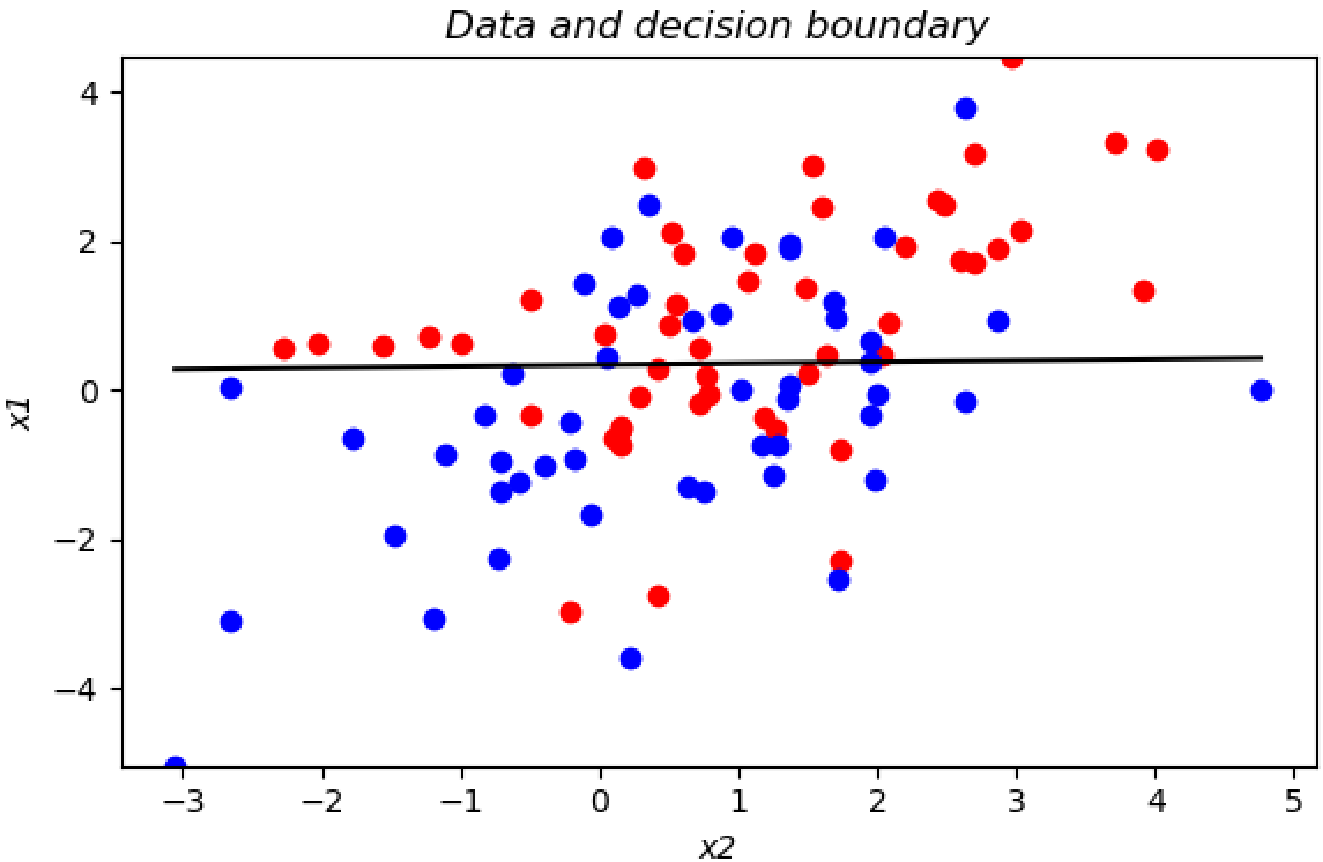
- **single layer perceptron** is the simplest type of artificial neural networks

- It can only classify linearly separable cases with a binary target (0 , 1).

How Single Layer Perceptron Works

- Single-layer perceptron will be trained to discriminate between these two classes.
- Each class is decribed by a mean vector and covariance matrix
- Defining the size of Training Dataset.
- Training with Gradient Descent Algorithm.
- Output of the Trained Data.

Discrimination of Two Classes



Parameter Test

- Learning Speed - 0.6
- Iteration / epoch - An **iteration** is a measure of the number of times all of the training vectors are used once to update the weights.
- Epochs defined in this experiment - 5
- Sigmoid Transfer Function - It was used between the hidden and output layers. For computing the variation in weight values between the hidden and output layers
- Sigmoid Function - 1 / (1 + exp(-x)) (Scales the value from 0 to 1)

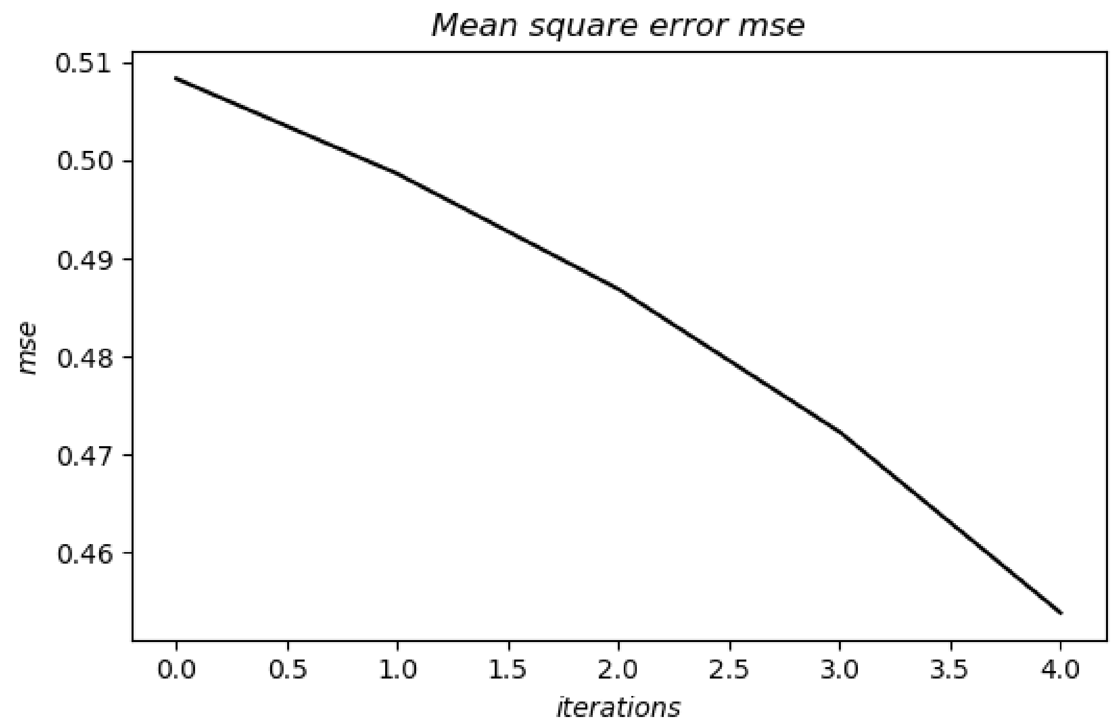
Training and Testing Samples

- Traning and Testing Samples = 50
- Class observed
- [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
- 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
- 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
- 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
- 1. 1. 1. 1.]
- Class predicted
- [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
- 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 0 0 1 1 0 1 1 1 0 1 1 1 1 0 1 1 0 1 1 1
- 0 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1 1]
- Training errors: 63
- Training errors: 63.0 %
- Class observed
- [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
- 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
- 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
- 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
- 1. 1. 1. 1.]
- Class predicted
- [1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
- 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1 1 1
- 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1 1 0]
- Test errors: 52
- Test errors: 52.0 %

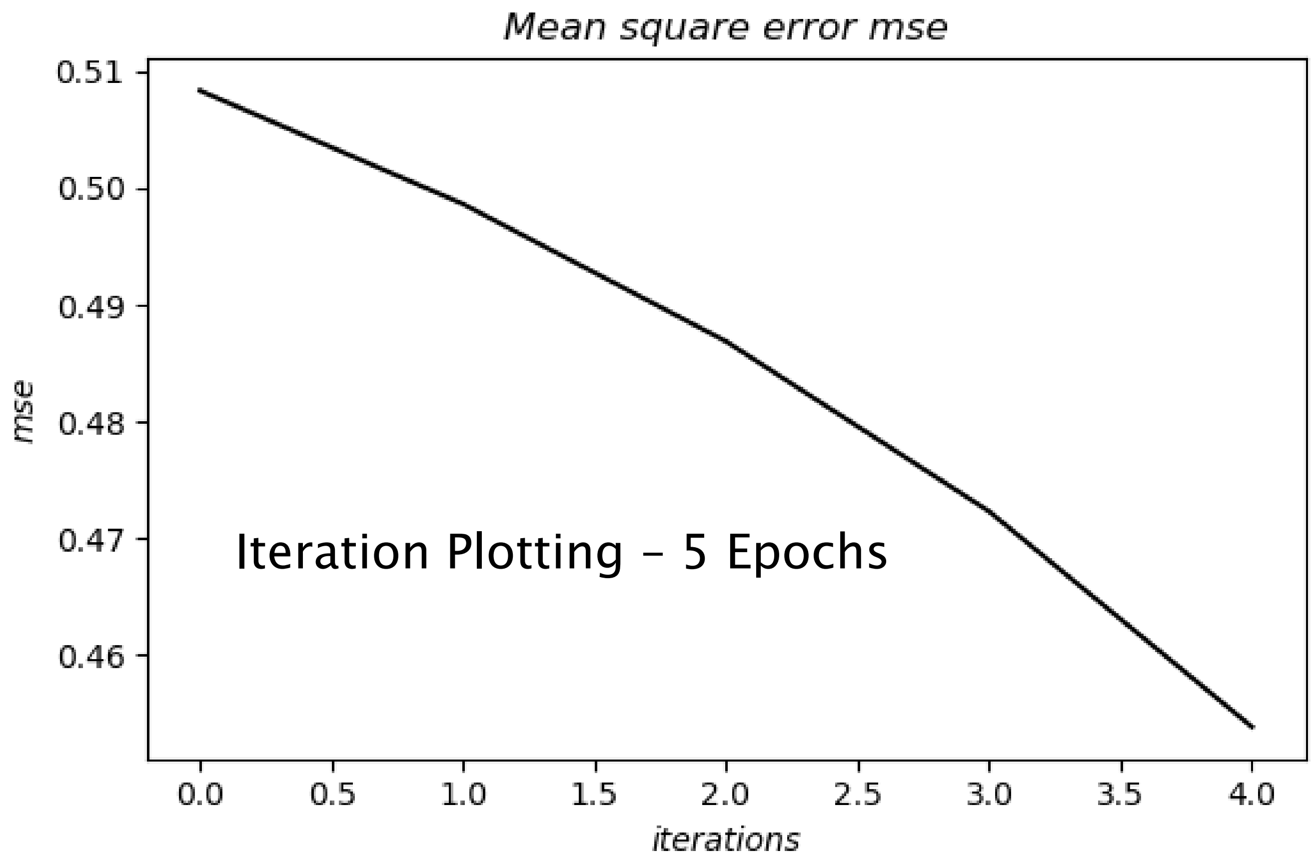
Calculating Mean Square Error

- MSE – Used for measuring the squared difference between target and actual output.
- mean1 = np.array([0.8, 1]) # mean vector class 1
- mean2 = np.array([0.5, 0]) # mean vector class 2
- var1=1 # variance of x1 feature
- var2=2
- var3=5# variance of x3 feature
- cor12=0.8

- MSE = sum((actual_output – self.target)**2)/N (i.e, Actual Output – Target)

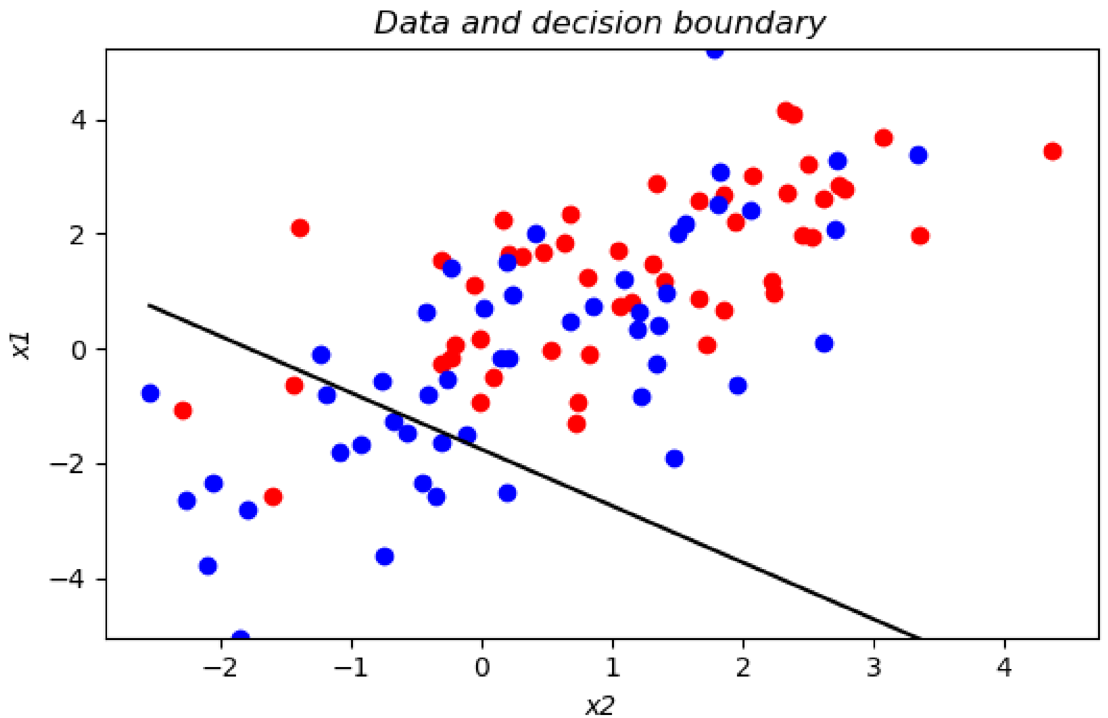


Error Classification After Each Iterations



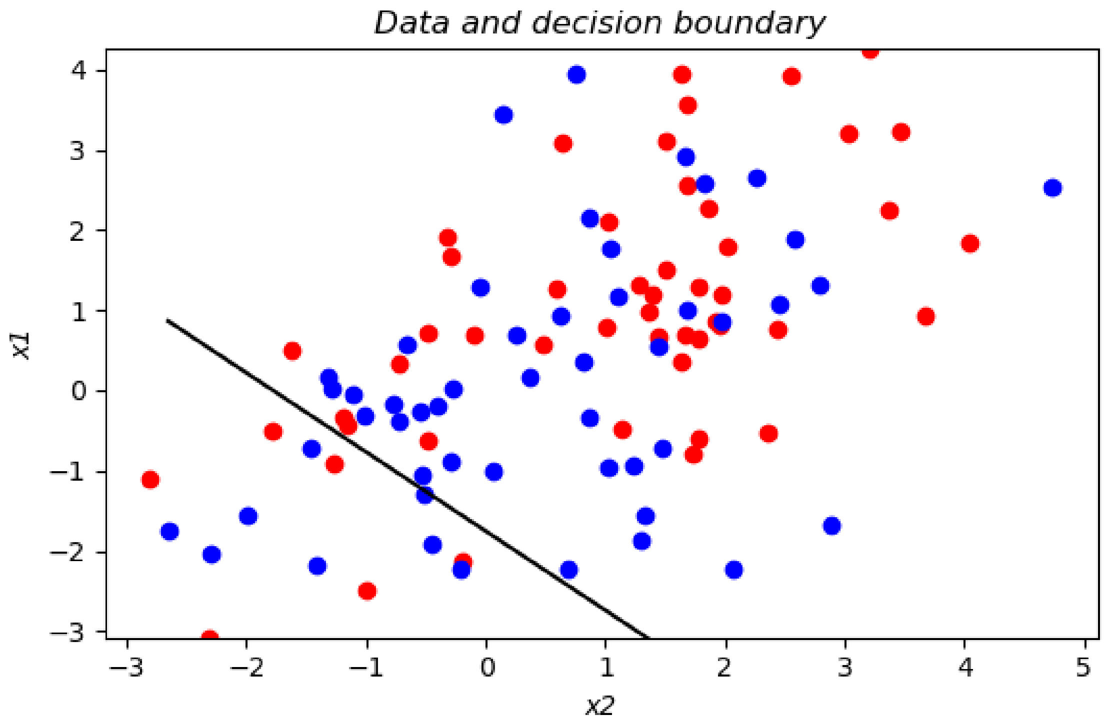
After Normalization, We can see that the error gradually decreasing

Testing Set – Classification Error



Classification Error – 63.0%

Testing Set – Classification Error



Classification Error – 52.0%

Distance Between The Classes

- mean1 = np.array([0.8, 1])
- mean2 = np.array([0.5, 0])
- var1=1
- var2=2

- cor12=0.8

- Calculating Co-variance = Corl2 * (Var1)^2 * (Var2)^2

- Mahalanobis Distance = (mean1 – mean2)^T . Covariance^(-1 or inverse) term. (mean1 – mean2) is the **distance** of the vector from the mean, then divide this by the covariance matrix.

Distance Output

- Mahalanobis distance between classes: 0.3