Project 2 Report

Classification Using Logistic Regression and Neural Networks

Name: Warren Mendonca

UBIT Name : wmendonc

Student Number: 50134835

1. Overview

The objective of the project is to implement and evaluate classification algorithms. I have implemented two different methods of classification namely:

(a) Logistic Regression (LR)
(b) Neural Network (NN)

Also I have compared their performance with a publicly available Classification Package for Neural Networks.

2. Training and Testing:

In this project, we are given images of the digits from 0 to 9. We use the extracted features from these images to learn to classify the digits. The extracted features are divided into two parts namely the training set and the test set. There are a total of 19978 records. Each record contains 512 features.

a. Training

The input of the training programs is an X = N *(D+1) matrix, where N is the number of training samples and D + 1 is the length of each training sample vector consisting of D features and the corresponding classification label. Using these inputs to train the data we find a weight matrix which we use on the test set to obtain the predicted class labels.

b. Testing

The testing programs should takes a Y = N'* D matrix as input and outputs a T = N' * 1 vector of classification labels. In testing phase N' is the size of the testing data set.

3. Methods used:

a. Logistic Regression:

We use logistic regression in multiclass classification problems. Multiclass logistic regression is used to train the network using all records. He we use the activation function and the trained weights to classify the digits here. The activation function we use here is the softmax function.

The expression for the softmax function is given below:

$$p(C_k|\emptyset) = y_k(\emptyset) = \frac{\exp a_k}{\sum_j \exp a_j}$$

We use the negative log likelihood error function to calculate the error. The function is given by:

$$E(\mathbf{w}_1,\ldots,\mathbf{w}_K) = -\ln p(T \mid \mathbf{w}_1,..,\mathbf{w}_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

We use gradient descent to train the program and obtain the weights. In our case we use the X as phi directly. The weights are updated in every iteration. We run the program for various iteration values given by the 'iter' variable to obtain a gradient that yields the least Error rate.

The gradient error with respect to weights is given by:

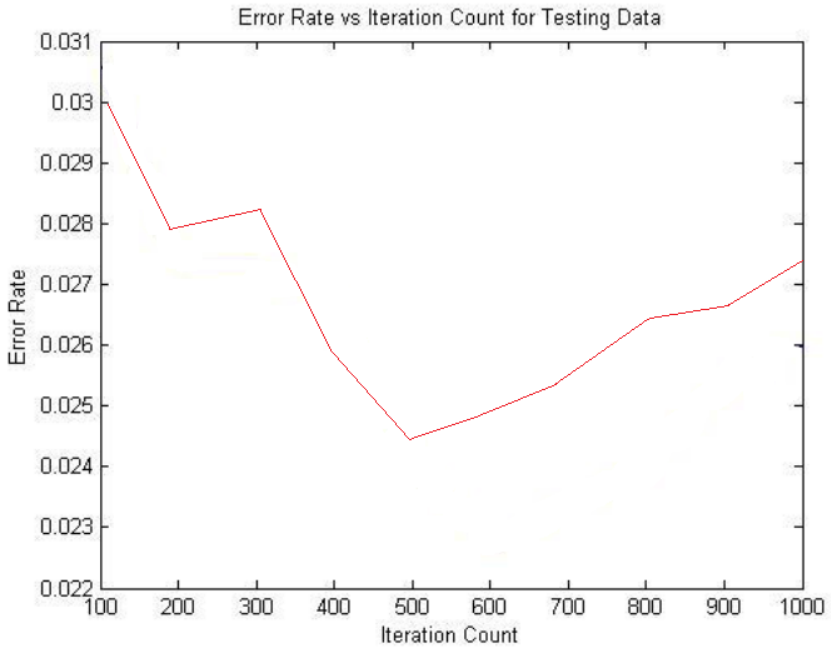
$$\nabla_{\mathbf{w}_j} E(\mathbf{w}_1,\ldots,\mathbf{w}_K) = -\sum_{n=1}^{N} \underbrace{(y_{nj} - t_{nj})\phi_n}_{\text{Error x Feature Vector}}$$

Results:

Iterations	Error (percentage)	Reciprocal rank (percentage)
500	2.43%	97.63%

The minimum error rate was achieved at the point here training rate given by the variable 'train' was set to 0.001. The optimum value for iterations was achieved at 500 iterations.

The below graph displays the results obtained for all iteration values and the resulting error rates achieved.



Graph of Error vs the Iterations

b. Neural Networks:

In the Neural network model we use the back propagation algorithm to obtain the derivatives. The data is trained using the using online gradient descent method. The weights are similarly updated in every iteration to give us the optimum value. We obtain two weights which are then used with the test data to obtain class labels for the data given. We use one hidden layer for the network.

Sigmoid activation functions are used for hidden layer and softmax activation functions are used for output layer.

The back propagation formula for this case is given below:

$$\Delta w_{hl}^{(2)} = \eta \sum_{p} \left(targ_l - out_l^{(2)} \right) out_h^{(1)}$$

$$\Delta w_{hl}^{(1)} = \eta \sum_{p} \sum_{k} \left(targ_k - out_k^{(2)} \right) w_{lk}^{(2)} . out_l^{(1)} . \left(1 - out_l^{(1)} \right) . in_h$$

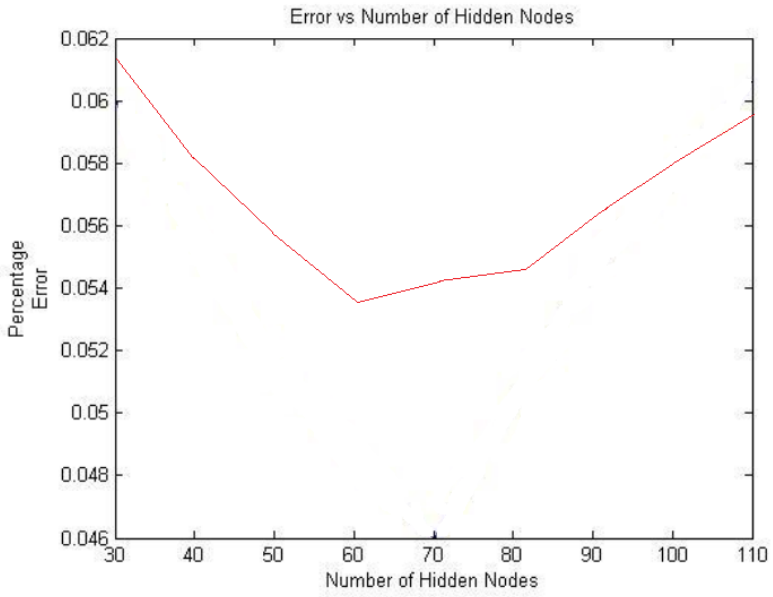
The parameters, number of hidden nodes and learning rate are selected based on graphs between the respective parameters and the error values.

I also varied the amount of hidden nodes and error to obtain the best values for hidden nodes. I have also kept the learning rate the same while calculating error values for updating the gradients in each of the layers.

Results:

Nodes	Error (percentage)	Reciprocal Rank (percentage)
60	5.32	96.98

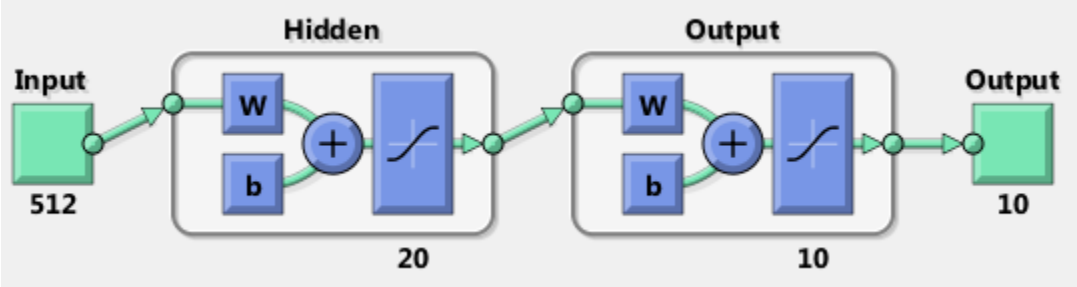
The least percentage error was achieved at fixing the number of hidden nodes at 60 with the error being at 5.32. Training rate was fixed at 0.02.



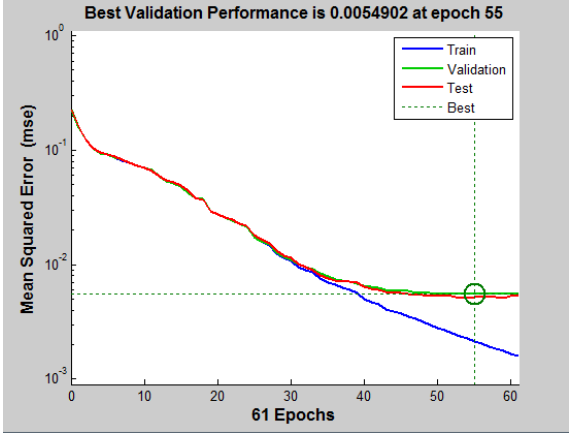
c. MATLAB Neural Network Toolbox:

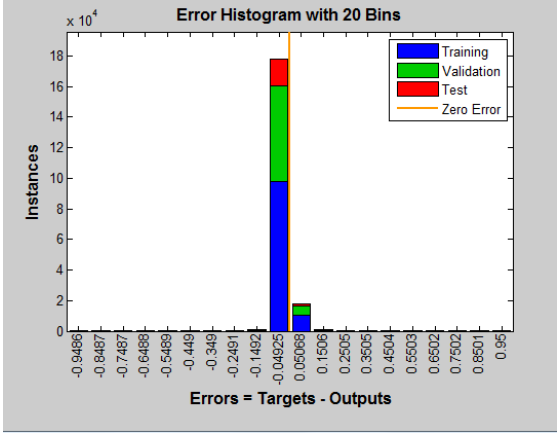
Matlab's neural network toolbox is used to do the classification. The link is http://www.mathworks.com/help/nnet/index.html.

The model parameters are displayed below in the diagram:

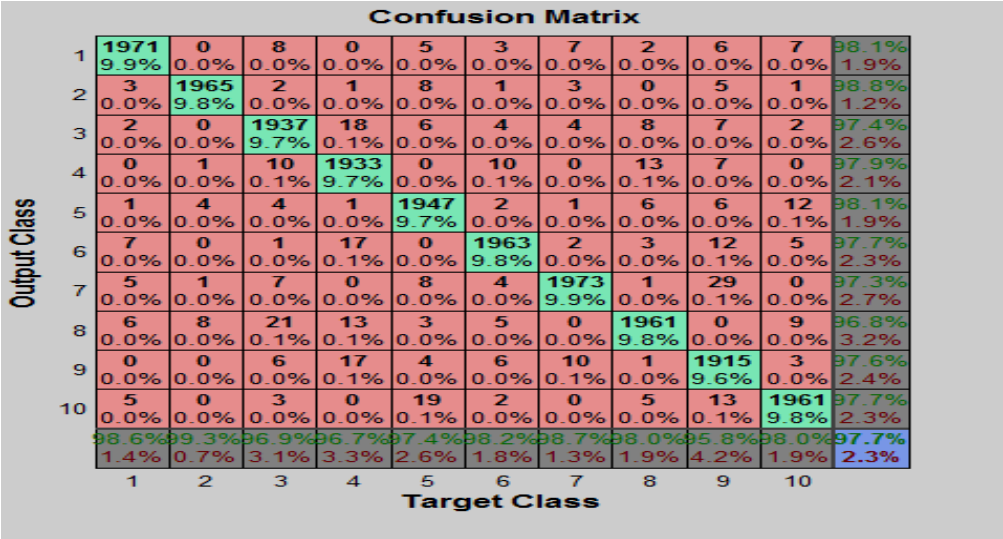


The below graphs display the best validation performance achieved and the histogram plot for the error:





The below graph plots the Confusion matrix achieved for the neural network:



From the matrix we can see that the error percentage is 2.3% for the data we have provided as input. The highest error was observed for the digit 8 and the least was observed for the digit 1. The model classifies the rest of the digits within an acceptable range of each other.