TBMI26 – Computer Assignment Report Supervised Learning

Deadline – March 14 2021

Authors: Shwetha Vandagadde Chandramouly(shwva184) and Suhani Ariga(suhar073)

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. Please upload the document in PDF format. **You will also need to upload all code in .m-file format**. We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the lab part of the course reported in LADOK together with the exam. If not, you'll get the lab part reported during the re-exam period.

1. **Give an overview of the four datasets from a machine learning perspective. Consider if you need linear or non-linear classifiers etc.**
    - Dataset 1 is linearly separable with two classes.
    - Dataset 2 also has 2 classes but is not linearly separable.
    - Dataset 3 has 3 classes and is not linearly seperable
    - Dataset 4 has 64 features and 10 labels (0 to 9). This is complicated and non-linearly separable.

2. **Explain why the down sampling of the OCR data (done as pre-processing) result in a more robust feature representation. See http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits**
   Down sampling of the OCR data result in more robust feature representation because this makes classifier less sensitive to small variations in data when represented in lower resolution, ie to say when down sampled to lower resolution, data no longer represents the minute variations.

3. **Give a short summary of how you implemented the kNN algorithm.**
    - Consider each datapoint to be classified one at a time , for each of them calculate Euclidean distance from point to be classified to all the points in training data.
    - Sort the training data points with respect to ascending order of their distance and pick out the top K data points as best neighbors.(K is determined by crossvalidation)
    - Now in a table take the class labels and votes for them in the best neighbors and the total distance of datapoints to the point to be classified of a particular class.
    - Pick the top voted class and label the datapoint to it, if there is tie, pick the class label with shortest distance.
    - repeat these steps for all datapoints in test data

4. **Explain how you handle draws in kNN, e.g. with two classes (k = 2)?**

For ties in kNN , we are first sorting on the class , an if there are ties , we are choosing the class with least distance as the label.

```
%now we will sort descending on votes , bringing the top voted classes
%above , next we sort ascendingly on distance to break the ties that
%might occur
count = sortrows(count,[-2,3]);
LPred(i) = count(1,1);
```

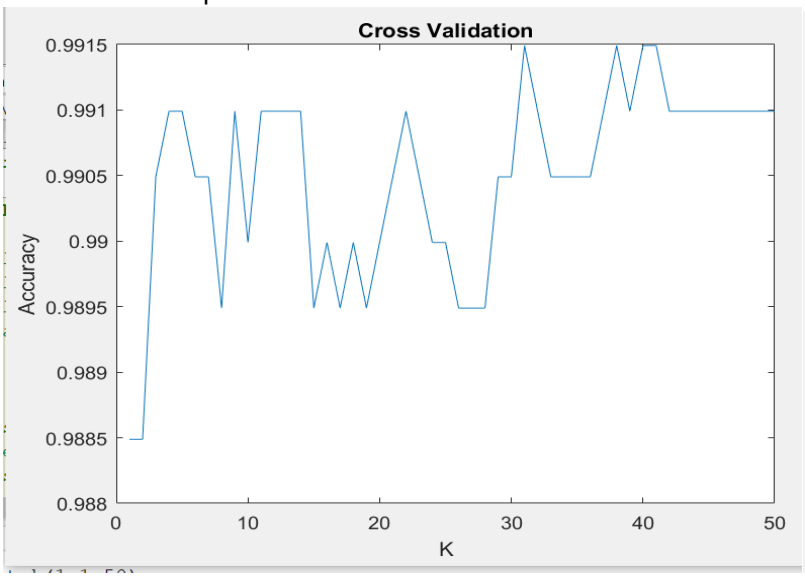
2 column of count contains the votes for each class , and the 3rd column contains the total distance

5. **Explain how you selected the best k for each dataset using cross validation. Include the accuracy and images of your results for each dataset.**

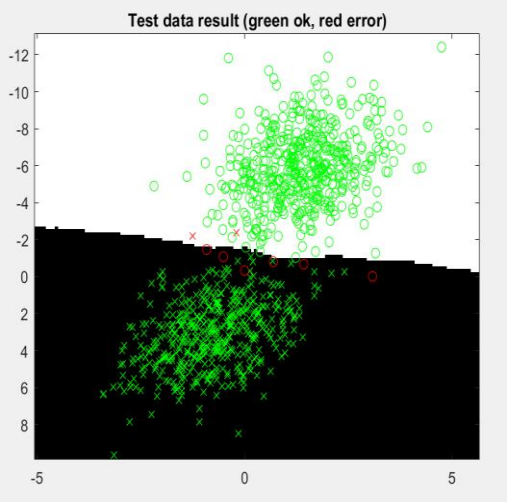
We used cross validation to choose the best k in cross_val_best_k.m . Here we have parted out data into 3 parts and followed the below steps to find accuracy.
1 : Accuracy 1 calculated on training data consisting of part 1 and 2 and test data as part 3.
2 : Accuracy 2 calculated on training data consisting of part 2 and 3 and test data as part 1.
3 : Accuracy 3 calculated on training data consisting of part 3 and 1 and test data as part 2.
Accuracy = accuracy1 + accuracy 2 + accuracy 3 / 3

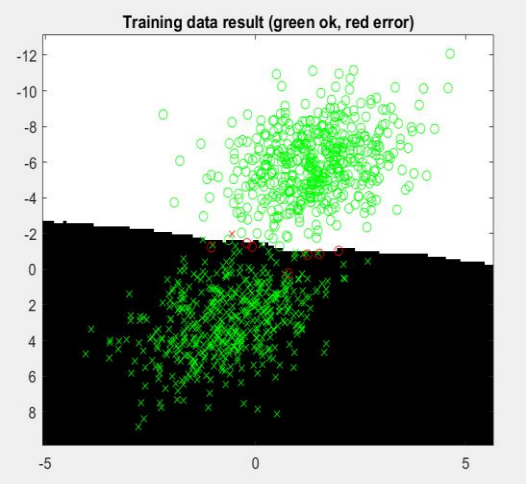
The above calculations are done for choice of k from 1 : 50 and the following results are obtained.

For Dataset 1 : Optimal K = 31

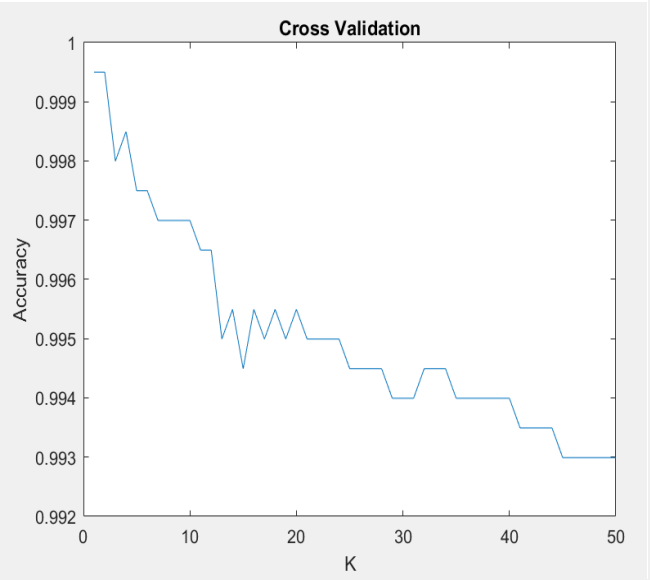



```
>> main_kNN()

cM =

   498     6
     2   494


acc =

    0.9920
```

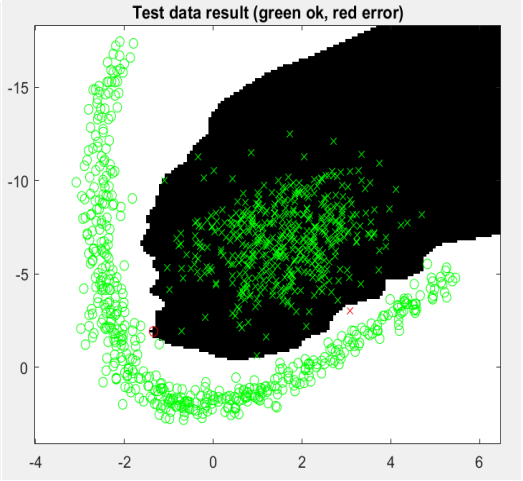




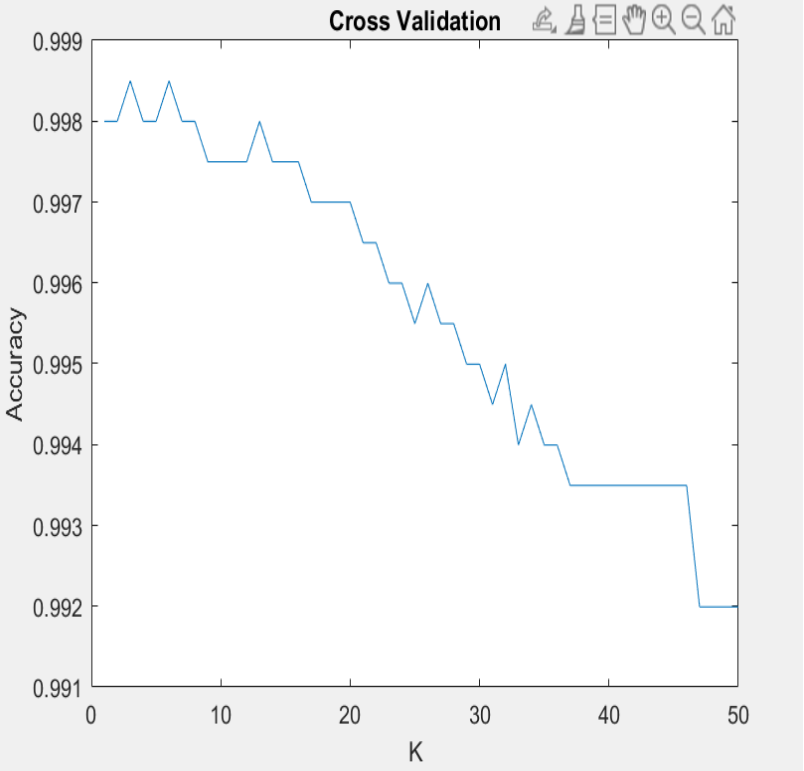
Dataset 2 : K = 1 and 2



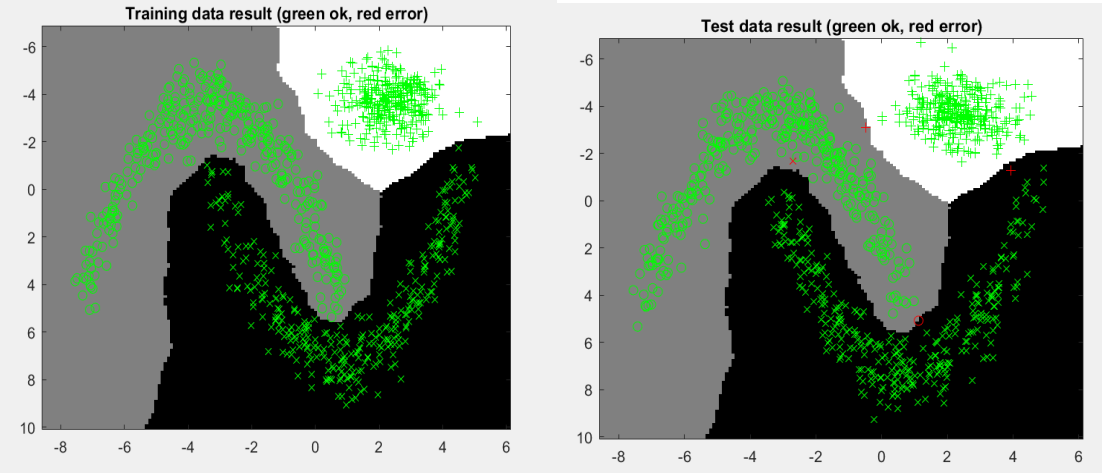

```
K>> main_kNN()

cM =

   499     1
     1   499


acc =

    0.9980
```



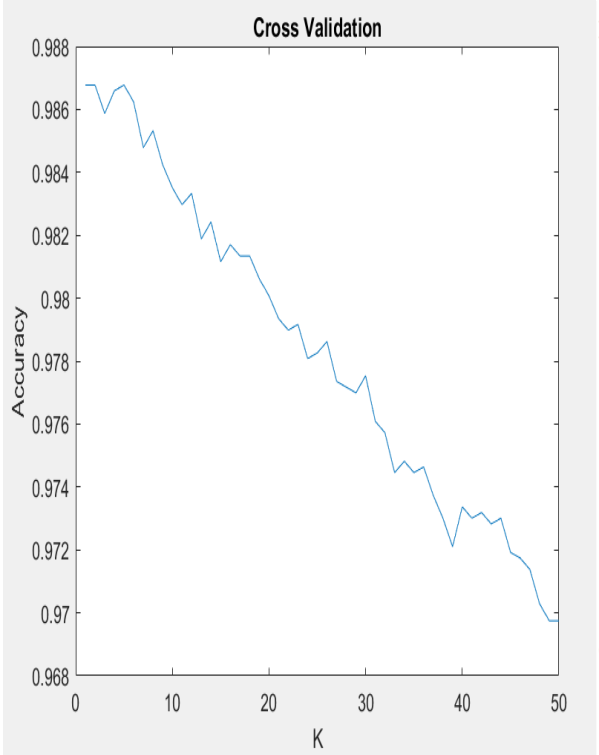
Dataset 3 : K = 3 or 6



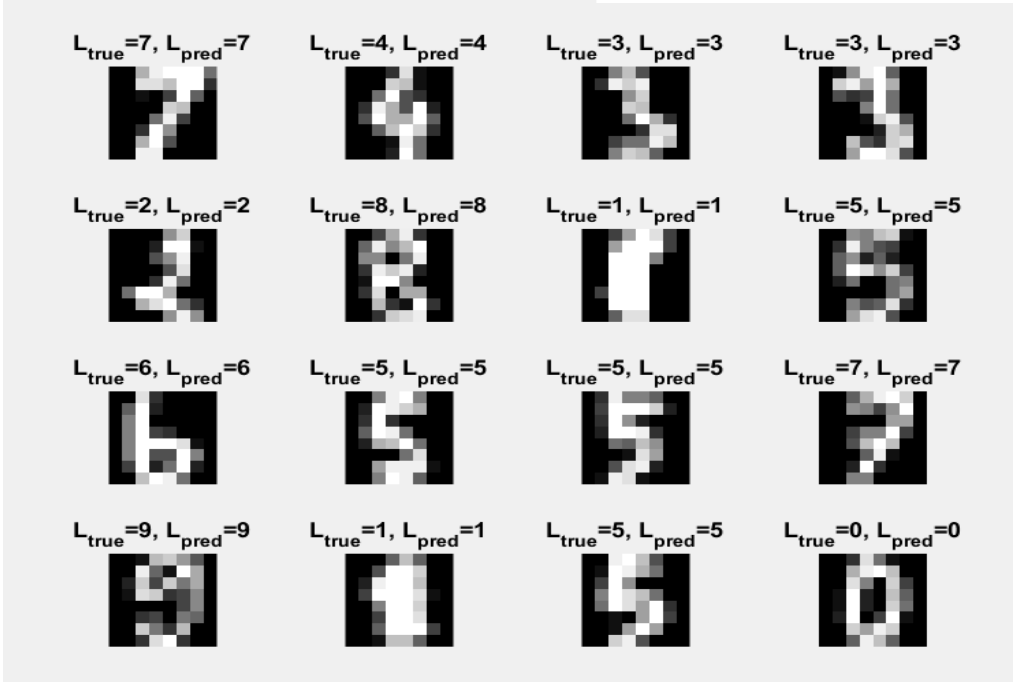

```
K>> main_kNN()

cM =

   332     1     1
     1   332     1
     0     0   331


acc =

    0.9960
```



Dataset 4 : K = 1 or 2




```
K>> main_kNN()

cM =

   277     0     0     0     0     0     0     0     0     0
     0   274     0     0     0     1     0     0    10     0
     0     0   277     1     0     0     0     0     1     0
     0     0     0   273     0     1     0     1     0     4
     0     0     0     0   275     0     0     1     0     4
     0     0     0     0     0   272     0     0     0     2
     0     0     0     0     1     0   277     0     1     0
     0     0     0     1     1     0     0   273     0     0
     0     1     0     0     0     0     0     0   263     1
     0     2     0     2     0     3     0     2     2   266


acc =

    0.9845
```



6. **Give a short summary of your backprop implementations (single + multi). You do not need to derive the update rules.**
   1. Initialize weights (each layer will have a set of weights) to random values close to 0
   2. Perform forward pass and calculate the output with initial weights
   3. Calculate the gradient, ie partial derivative with respect to weights(each layer will have a set of weights)
      For single layer we will have only one set of weights, the gradient of it is calculated as follows

```
grad_w = (2* XTrain' * (YTrain - DTrain))/ NTrain;
```

For multilayer , here we have two set of weights V and W

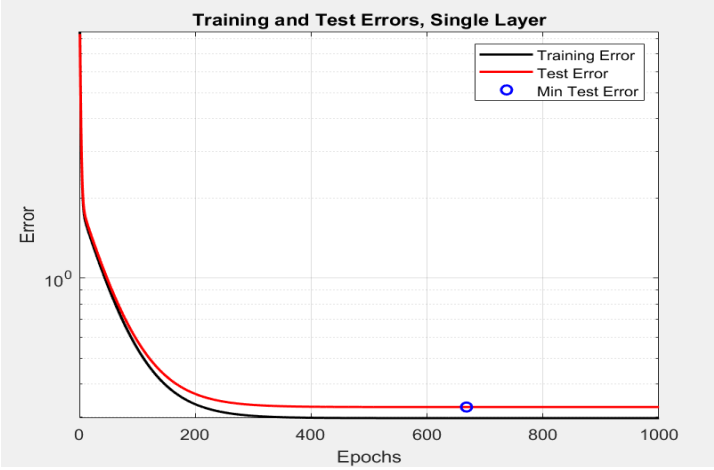
```
grad_v = (2 * HTrain' * (YTrain - DTrain))/(NTrain * NClasses);
% Gradient for the output layer
grad_w = (2 * XTrain' * ((YTrain - DTrain) * Vout( 1:(size(Vout,1)-1) ,:)' .* (1-HTrain(:,1:(size(HTrain,2)-1)).^2) )) / (NTrain * NClasses)
% And the input layer
```

4. Next we update the current weights via step function.
5. Calculate the errors for updated weights.
6. Repeat 2 through 5 for a given number of epochs(iterations)

7. **Present the results from the neural network training and how you reached the accuracy criteria for each dataset. Motivate your choice of network for each dataset. Explain how you selected good values for the learning rate, iterations and number of hidden neurons. Include images of your best result for each dataset, including parameters etc.**

Dataset 1 :
Dataset 1 can be classified efficiently with just single layer as this is already linearly separable, there is no need to add hidden layers to this.

For this we have chosen learning rate : 0.005 and number of iterations 1000.
As this is a linearly separable dataset, we have set number of iterations 1000, but we can see that it converges early (around 630)
We get accuracy of 99%.



Dataset 2 :

This dataset is not linearly separable and has 2 classes. This is complicated that dataset 1 and will need hidden layers as it is not linearly separable in input space.

We have kept 5 hidden neurons (higher number of layers as this is complex than previous dataset)

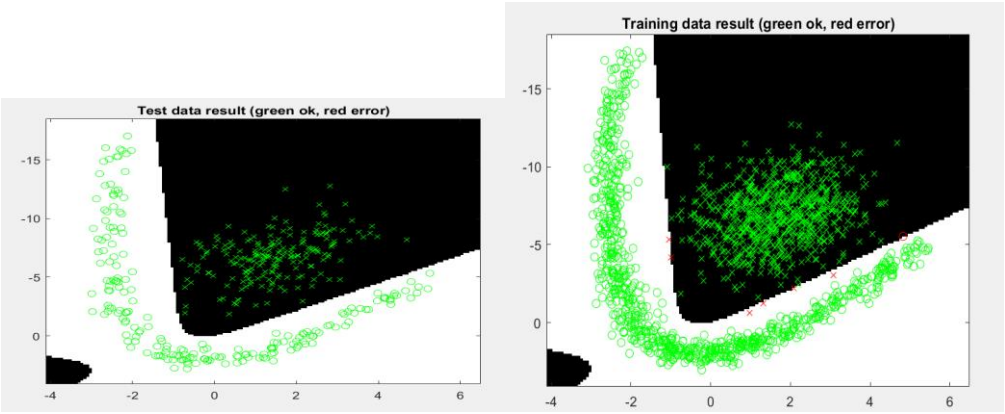
10000 number of iterations (We actually get 99% accuracy with just 4000 to 5000 epochs , but we have kept 10000 to show the min test error in the graph). Learning rate of 0.01 (we have increased the learning rate as there is enough space between clusters)

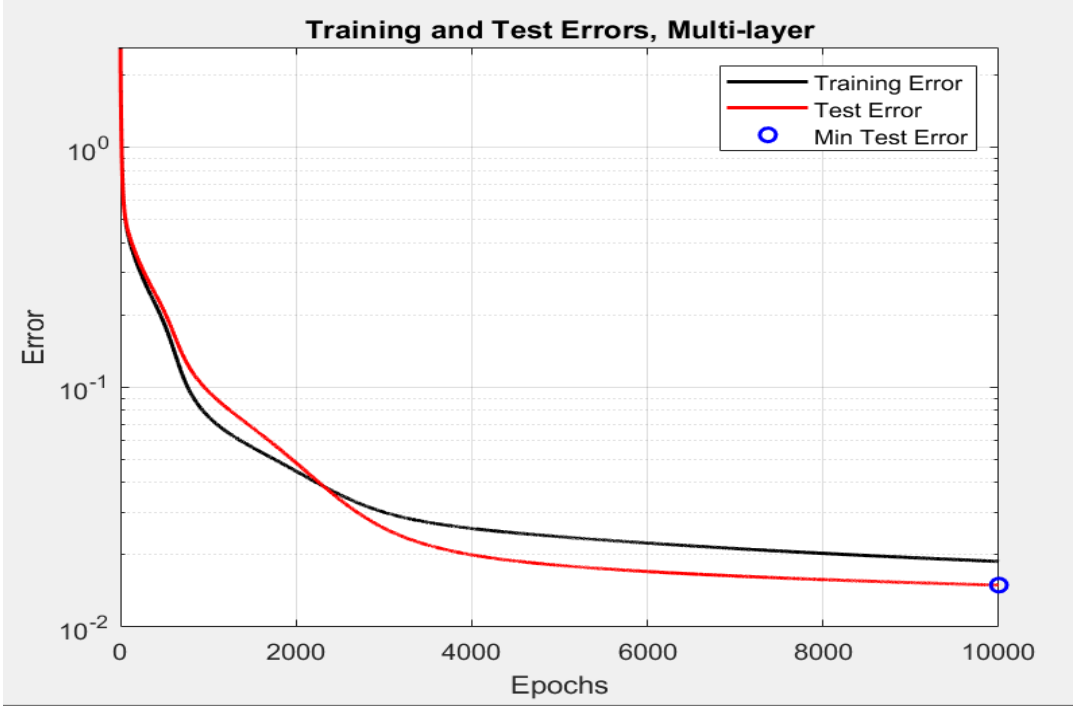
We have obtained the following results:

Time spent training: 3.3823 sec

Time spent classifying 1 sample: 3.2015e-07 sec

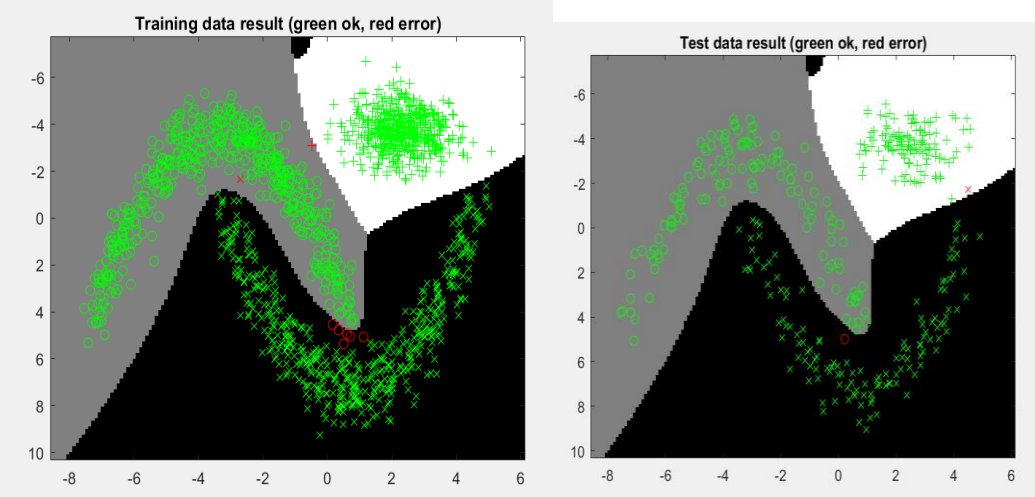
Test accuracy: 1

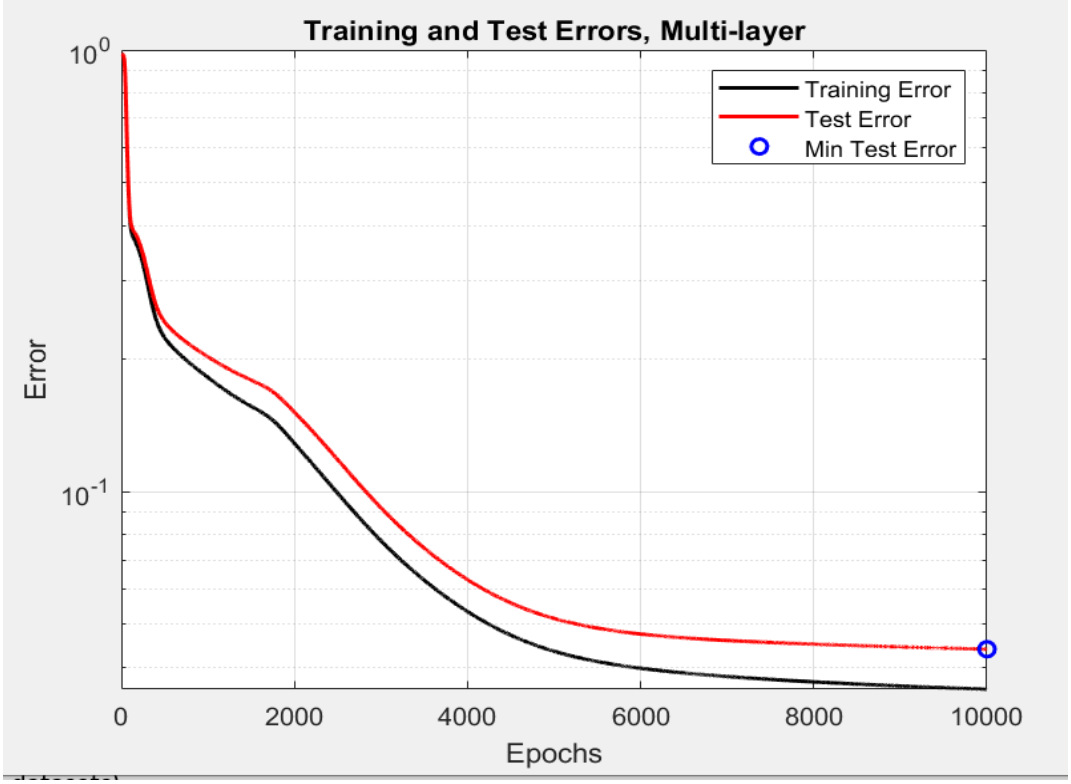




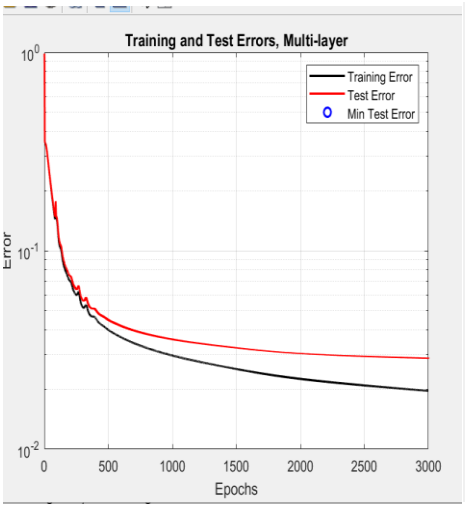
Dataset 3 :
This dataset is not linearly separable and has 3 classes. This is complicated that dataset 1 & 2 and will need hidden layers as it is not linearly separable in input space.
We have kept 9 hidden neurons (higher number of layers as this is complex than previous datasets)
10000 number of iterations(this is the required number of epochs to obtain min test error).
Learning rate of 0.01.
We have obtained following results

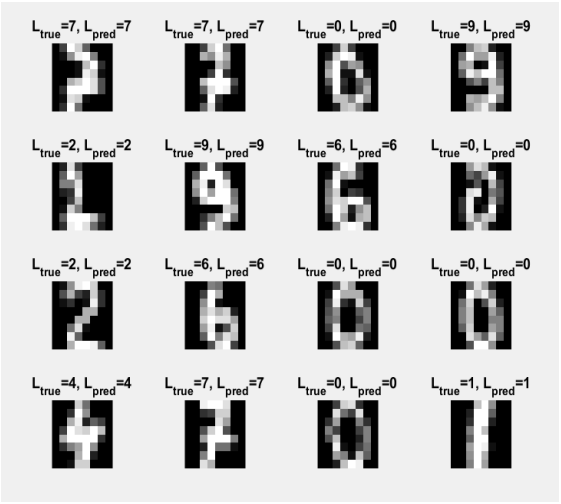
Time spent training: 10.7686 sec
Time spent classifying 1 sample: 5.9053e-07 sec
Test accuracy: 0.99499





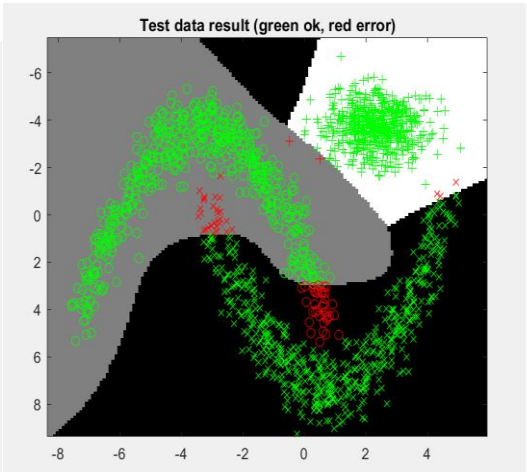
Dataset 4 :
This dataset is not linearly separable and has 10 classes. This is complicated that dataset 1 2 3 as there are many (64) number of features and will need many hidden layers as it seems to have a complex relationship among the features.
We have kept 57 hidden neurons (higher number of layers as this is complex than the previous datasets)
3000 number of iterations (Increasing the number of neurons and epochs further will lead to better results, but this gets computationally heavy and only 96% of accuracy result was expected to produce , hence we have chosen the above values). Learning rate of 0.008.
We have obtained following results.
Time spent training: 71.2506 sec
Time spent classifying 1 sample: 2.226e-06 sec
Test accuracy: 0.97455





8. **Present the results, including images, of your example of a non-generalizable backprop solution. Explain why this example is non-generalizable.**
   Non generalizable error means we get good performance on training data but however this solution does not generalize well , ie to say it does not perform well with test data.

   For the example we have considered dataset 3 , we had achieved 99.5 % accuracy as optimal solution for this.
   Now we are using the epochs : 10000, hidden neurons: just 4 and same learning rate , but we are training only on 1/10 of the data and testing on the rest. The performance improves and looks looks pretty good on training data , but does not perform well on test data.



9. **Give a final discussion and conclusion where you explain the differences between the performances of the different classifiers. Pros and cons etc.**

kNN algorithm:

- Performs well on both linear and non-linearly separable data.
- Not good for large data, as it needs to store all the data points , hence will be slow.
- Classification takes long time.
- Need to pick a good K value.

Single Layer:

- Faster than multilayer network.
- Works well with linearly separable data.
- Performs poorly with non-linearly separable data.

MultiLayer :

- Works well with non-linearly separable data as well.
- Unlike kNN , final model doesn't take up a lot of storage.
- Computationally heavy , hence takes long time depending on the number of hidden neurons and epocs.
- Need to pick a suitable activation function, learning rate, initial weights, number of neurons and number of iterations.

10. **Do you think there is something that can improve the results? Pre-processing, algorithm-wise etc.**

May be trying out different activation functions would help us improve the accuracy and reduce the run time. It would also be better if we could incorporate a smarter way to initialize the weights, this would help us reduce run time as well as helps us from not getting stuck in local minima. Here we have just tried using one hidden layer with number of

neurons, may be increasing the number of hidden layers would help us get better results.