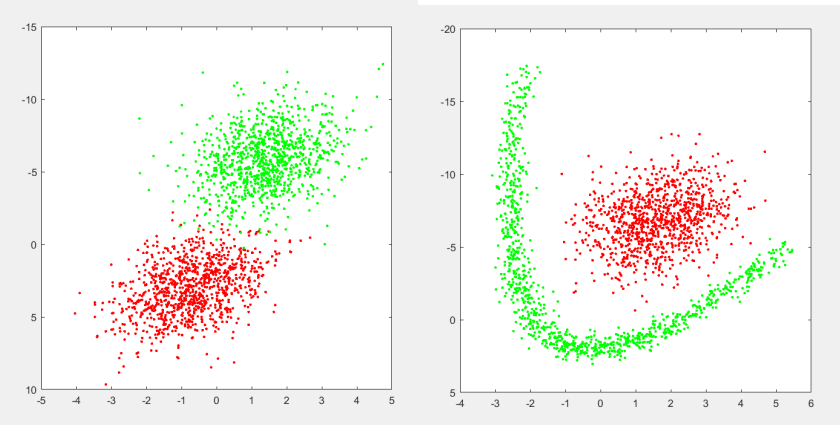
TBMI26 – Computer Assignment Report Supervised Learning

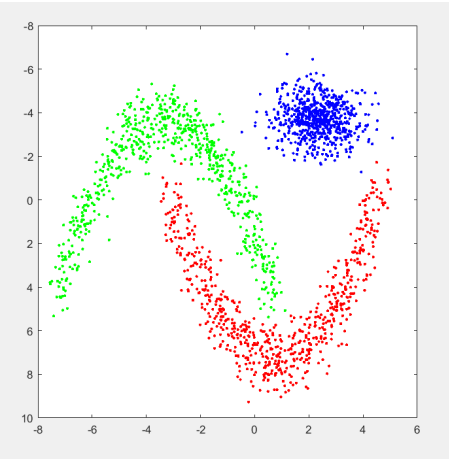
Deadline – March 14 2021

Author/-s:
Tim Yuki Washio (timwa902)
Nicolas Taba (nicta839)

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.**
   The first 3 datasets represent a points that have two attributes and are assigned to two or three classes (dataset3). The fourth dataset is handwritten digits with 64 features and 10 classes.
   The only dataset that can be separated linearly is the first dataset. Some of the points do overlap with the other class, but the performance of a linear classifier should be good enough. The other dataset cannot be separated by a linear classifier.The datasets 1 to 3 are shown below in order.





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**
   The data is reduced in dimensions to 8x8 bitmaps that can then in turn be represented as 64bits. This dimensional reduction helps standardize all the data, reduce the number of features to be studied and saves computational resources. This pre-processing reduces the burdens of differences in resolution and size that can occur between different elements of the dataset.

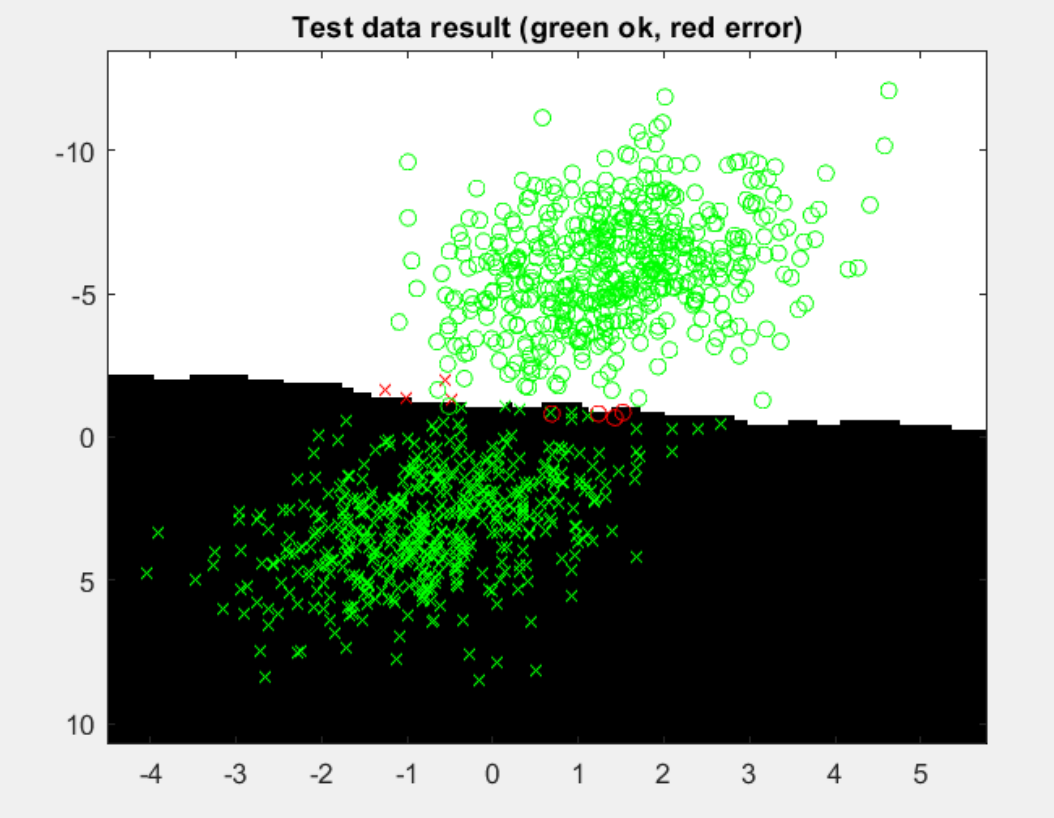
3. **Give a short summary of how you implemented the kNN algorithm.**
   In our implementation of the kNN algorithm, we calculate the Euclidean distance between a point and all others and sort them by distance and keeping track of the index. We then tabulate the k-first elements and find the label that is the most represented. The label that is the most represented gives the label of our point.

4. **Explain how you handle draws in kNN, e.g. with two classes (k = 2)?**
   For draws we choose the label of the nearest neighbour. This implementation works well enough for the datasets that we have and we did not foresee any issues nor encountered any. A different implementation for solving the draw could be to increase or decrease k by 1 to solve for a particular point.

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.**
   To find the optimal k value we shuffle our training data first since we know that our dataset is sorted. We then separate our training data into N chunks of the same size (depending on which dataset we are using we chose 9 or 10). We then iterate over different values for k (1,2,

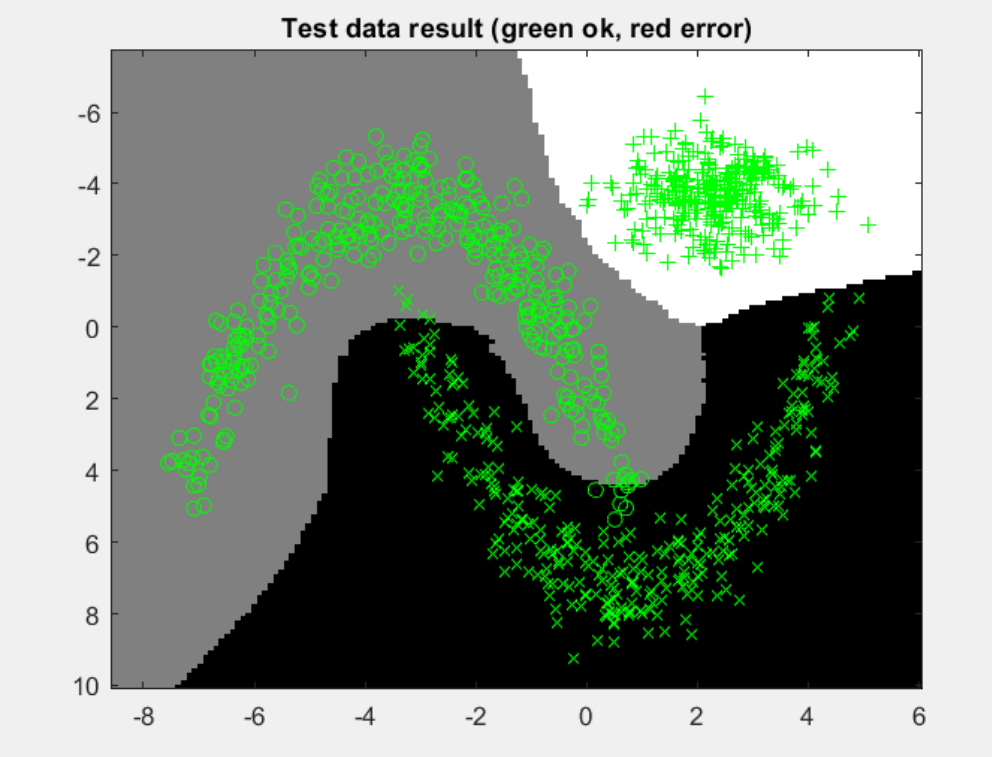
…,50) and use N-fold crossvalidation to find different accuracy values for N-1 training chunks and 1 Test chunk. In the end we find the average over all folds and k values. We then select the optimal k by choosing the maximum average accuracy found.

Dataset 1:
Accuracy: 0.992
Optimal K: 6

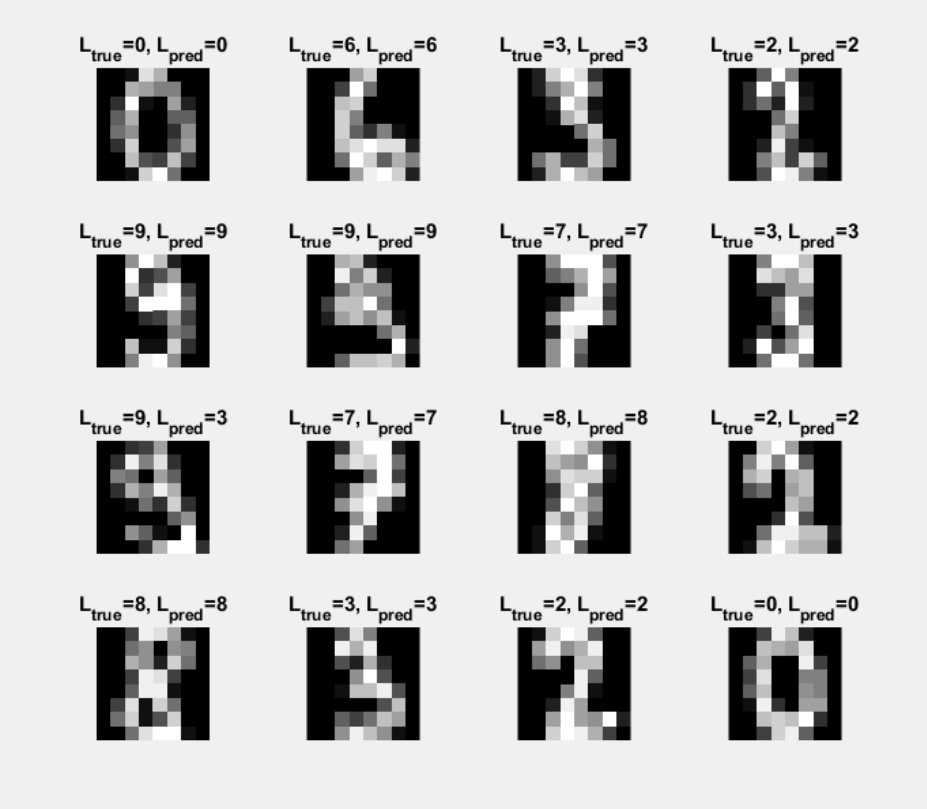


Dataset 2:
Accuracy: 1
Optimal K: 1

Dataset 3:
Accuracy: 1
Optimal K: 6



Dataset 4:
Accuracy: 0.9856
Optimal K: 3



6. **Give a short summary of your backprop implementations (single + multi). You do not need to derive the update rules.**
   For both the single layer and multilayer implementation we start by initializing the data and separating it into training and test data. We add a bias to both. We then initialize the weights as random numbers between 1 and 0.

   In the case of the single layer network. Our objective with the training data is to try to find the set of weights that minimizes the misclassification error. We implement the calculation to find the gradient of the weights in order to update them.

$$\frac{2((X_t W) - D_t)X_t^T}{N}$$

   The learning rate then determines how much, in the direction of the gradient, our next guess for the updated weights will be.

   For the case of the multilayer, we also initialize the number of neurons in our hidden layers (there are 2 hidden layers). In a similar fashion to the single layer, we implement the calculation to find the updated weights for the first and the second layer and the learning rate determines how far we move in the direction of the calculated gradient for our next guess.

   For both cases, we iterate for the number of epochs that we determine heuristically.

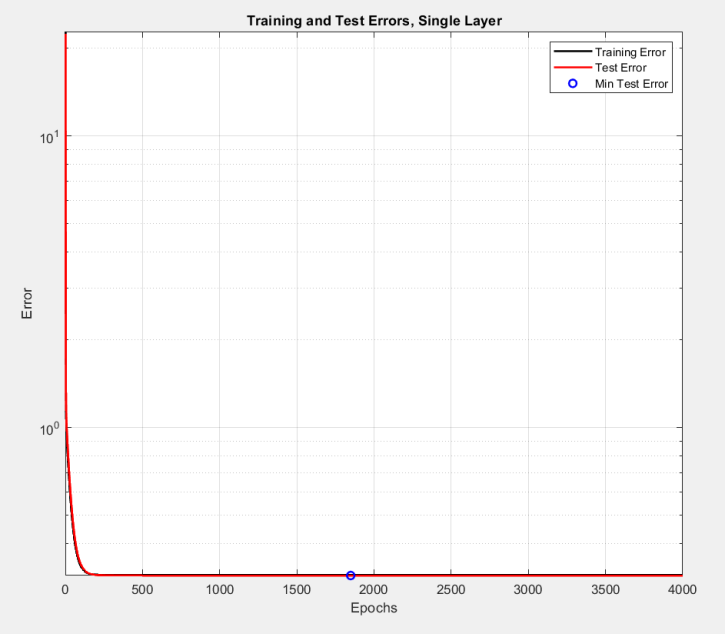
7. **Present the results from the neural network training and how you reached the accuracy criteria for each dataset. Motivate your choice of network for each dataset. Explain how you selected good values for the learning rate, iterations and number of hidden neurons. Include images of your best result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

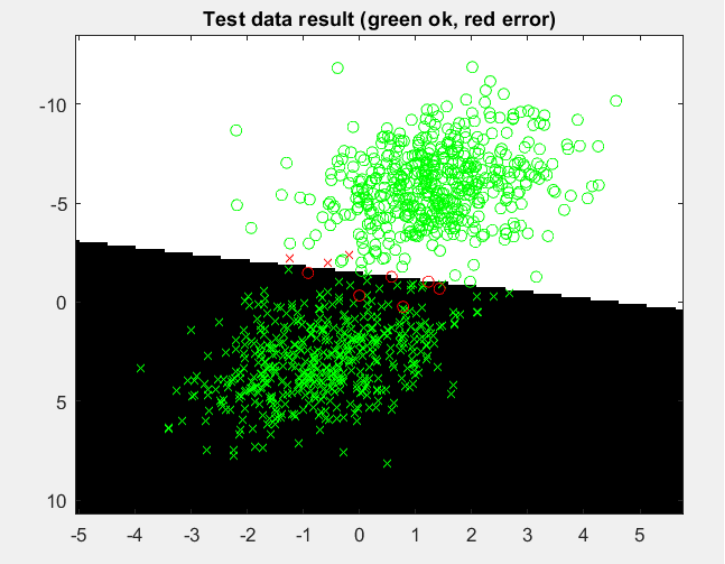
The values for learning rate, iterations and number of hidden neurons are chosen heuristically. However, there are certain rule of thumb that we can follow in order to make the exploration and the number of tries a little bit easier when experimenting.

For linearly separable labels for our data, we can use a simple single layer network to solve the classification as the problem can be reduced to linear regression. For classification with data that is not linearly separable, the rule of thumb that we follow is that the size of the hidden layer can usually be somewhere between the size of the input and the ouput layers. Longer iterations will lead to lower training errors and a higher learning rate will lead to faster convergence towards the optimal error. We try to keep in mind that a learning rate too large may lead us to get out of the local optima.

Using this set of simple rules, we perform the classification on the different datasets.

Dataset 1:
4000 iterations
Learning rate : 0.01
On single layer network

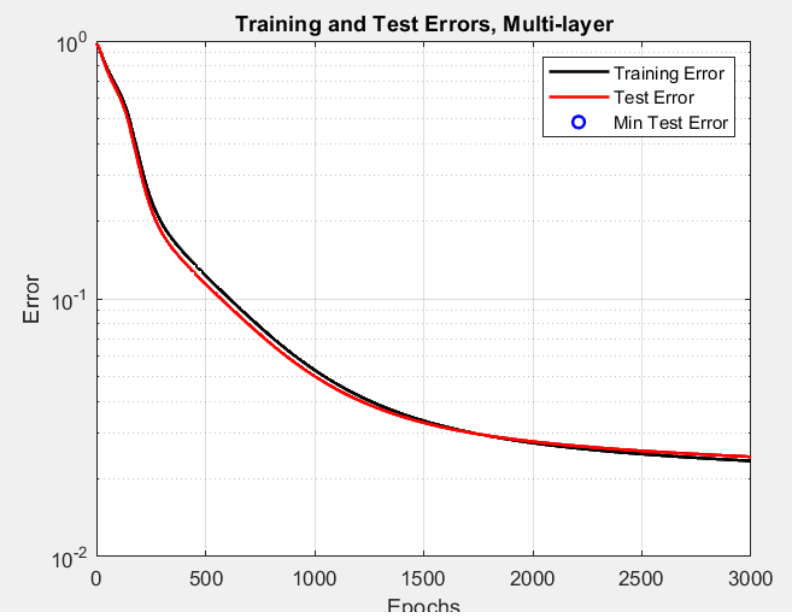





```
cM =

   497     6
     3   494

Time spent training: 0.1464 sec
Time spent classifying 1 sample: 2.3175e-07 sec
Test accuracy: 0.991
```

Note: We could have reduced the number of iteration steps, but 4000 seemed like a good number in order to keep the error low with a certain confidence.

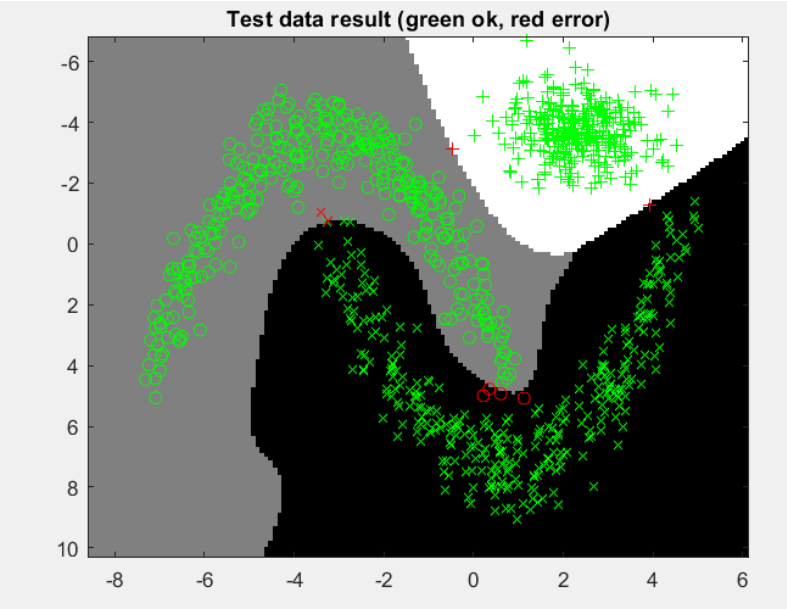
Dataset 2:
3000 iterations
Learning rate : 0.01
5 hidden neurons

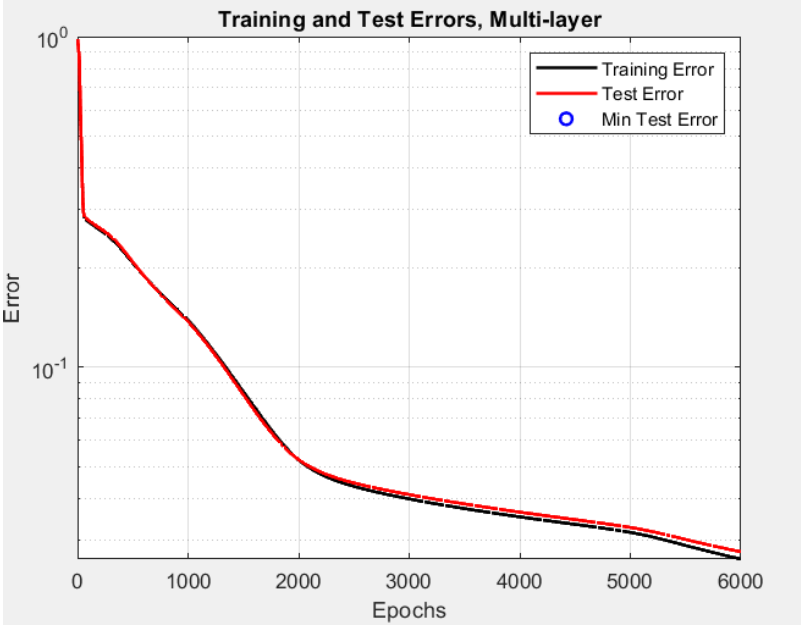



```
cM =

   332     0
     1   333

Time spent training: 0.88124 sec
Time spent classifying 1 sample: 6.2668e-07 sec
Test accuracy: 0.9985
```

Note: We did not gain many benefits from increasing the learning rate in this dataset. The operation time is still low enough to not have to justify more iterations.

Dataset 3:
6000 iterations
Learning rate: 0.02
20 hidden neurons



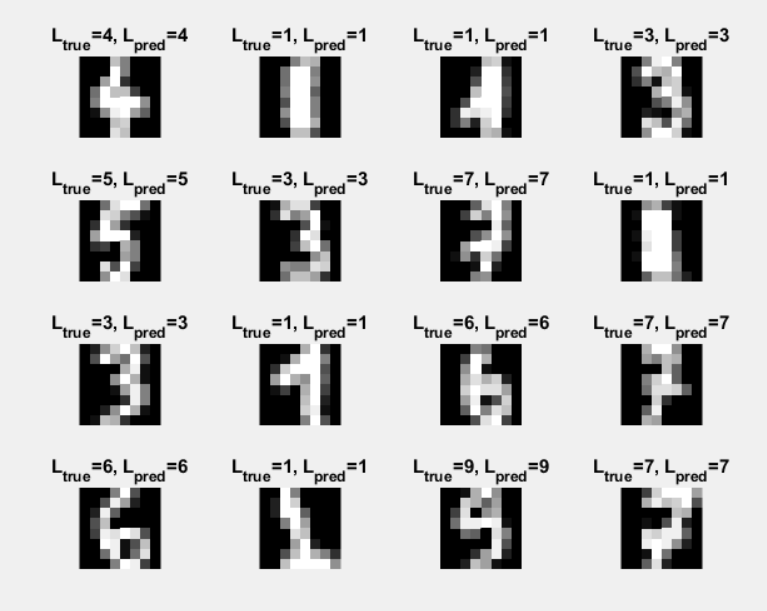


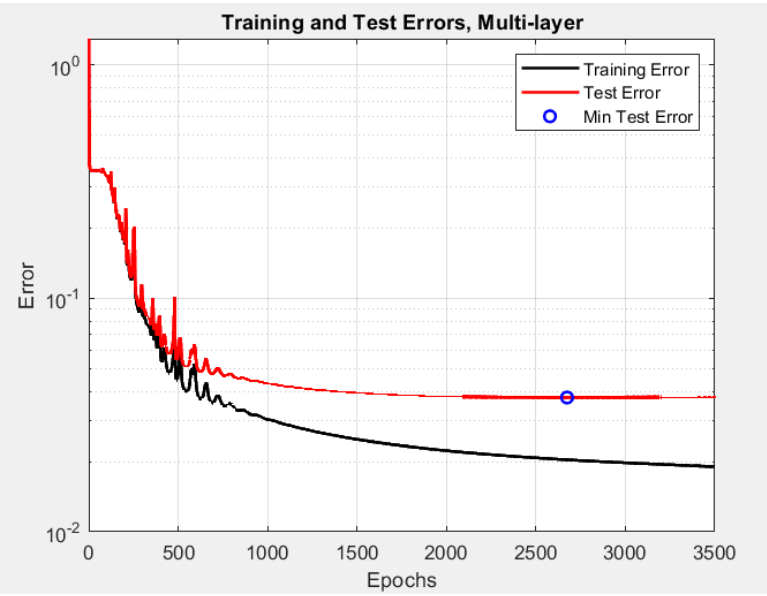

```
cM =

   331     4     1
     2   329     1
     0     0   331

Time spent training: 4.8302 sec
Time spent classifying 1 sample: 1.1523e-06 sec
Test accuracy: 0.99199
```

Note: Increasing the number of hidden neurons and iterations yielded the best improvement in performance. We tried to compensate by increasing the learning rate.

Dataset 4:
3500 iterations

Learning rate: 0.02
40 hidden neurons



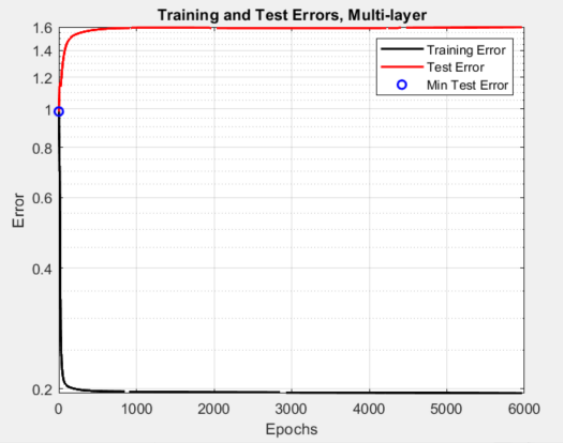



```
Time spent training: 27.6127 sec
Time spent classifying 1 sample: 2.1502e-06 sec
Test accuracy: 0.96606
```

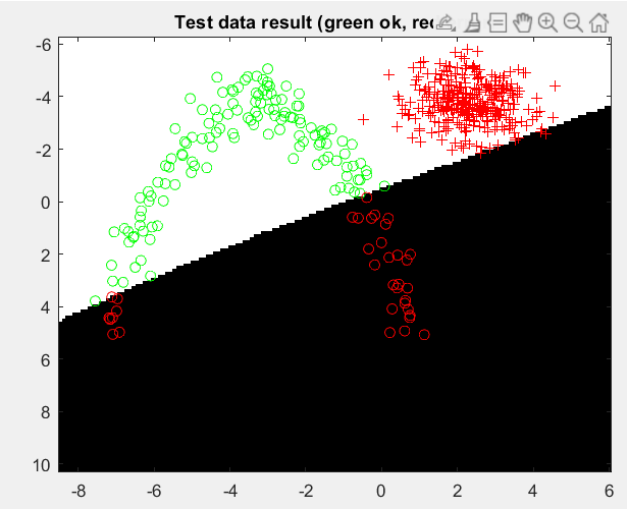
Note: The number of hidden neurons is increased a lot in order to conform more closely to the 64 features of the input layer. Performance was increased by reducing the number of epochs and keeping the learning rate high.

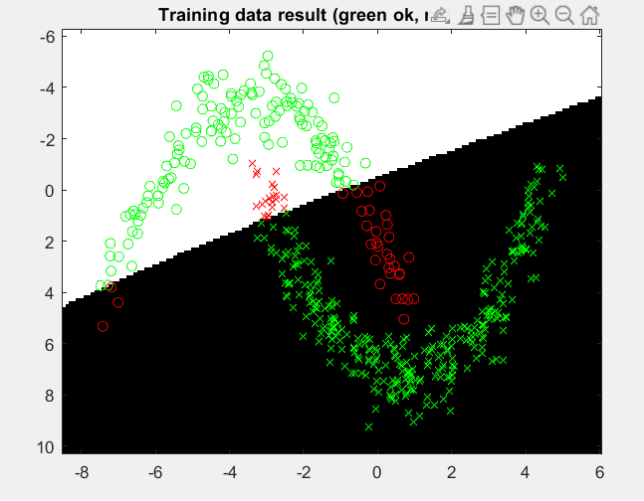
8. **Present the results, including images, of your example of a non-generalizable backprop solution. Explain why this example is non-generalizable.**
Non-generalizable solutions are solutions that do not perform well when encountering new data (test data). In order to present this property, we carefully choose the data such that we know that our solution will be non-generalizable. For our training data, we choose data

points that only have labels 1 and 2 and for the test data, we choose data points that only have labels 2 and 3. The result is the following:




```
cM =

     0    35     4
     0   133   329
     0     0     0

Time spent training: 2.3617 sec
Time spent classifying 1 sample: 3.6791e-06 sec
Test accuracy: 0.26547
```





From this data, we can see that the model performs very badly when encountering new data. We artificially chose the datasets such that there was one class that was previously unseen in the training phase of our neural network. This illustrates that supervised neural network cannot find new class labels if they do not encounter them before.

9. **Give a final discussion and conclusion where you explain the differences between the performances of the different classifiers. Pros and cons etc.**
   The advantage of using the kNN algorithm over neural networks is that it is a fast, understandable implementation of an algorithm that performs well compared to the neural network in this context and with these datasets. However, the drawback of the kNN algorithm is that all the data is stored in order to make a prediction, which in the case of very large dataset is very computationally intensive. This algorithm also faces the issue that finding an optimal k large enough for the calculations to be quick and generalizable, but small enough that it performs well is computationally intensive using cross-validation because we have to store all the data in order to compare. We should also keep in mind of the bias introduced by our decision of how to solve draws in the implementation. Small k will lead to models that are too complex even though it can perform well on non-linear problems. The kNN algorithm seems to be an easy go-to algorithm for exploratory models that can quickly yield insight into the data to be analyzed/classified.
   Neural networks require to be trained and can thus be more cumbersome to use compared to the kNN algorithm. The structure of a neural net can however easily be ported to another problem set and has thus more flexibility compared to the kNN algorithm. If we compare the neural networks between each other, we find that multilayer neural nets perform better than single-layer ones. Even if the single layer can theoretically solve any problem with enough neurons, it is much more computationally expensive to do so than to add more depth to the network. We should also add here that the weights of neural networks and the attributes are difficult to interpret which makes them difficult to share and explain.

10. **Do you think there is something that can improve the results? Pre-processing, algorithm-wise etc.**
    Algorithm wise, we have explained already that for the kNN algorithm, we could improve the resolution of draws by dynamically implementing a new k for the point that has a draw that would either be one more or one less in order to solve the draw and use an odd k.
    For the neural networks, the multilayer network could be improved by randomly switching off certain neurons in order for the network as a whole to not rely too much on a single attribute. This has the added advantage to yield a more generalizable solution. The increase in the number of hidden layers could also be implemented, but could add uneccasary complexity.
    In terms of pre-processing and for complex data like the hand-written digits, one could imagine a way of handing missing data and outliers statistically by implementing an probabilistic approach to the labeling. In the example of handwritten digits, if we did not know the true labels of the digits, we could use one-hot encoding and probabilities to assign labels to unlabeled data.

11. **Optional task (but very recommended). Simple gradient decent like what you have implemented can work well, but in most cases we can improve the weight update by using more sophisticated algorithms. Some of the most common optimization algorithms are summarized nicely here:**
    https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
    Implement one or a few different optimizers and compare the speed at which the training converges. A good starting point is to implement momentum gradient decent.