# TBMI26: A1

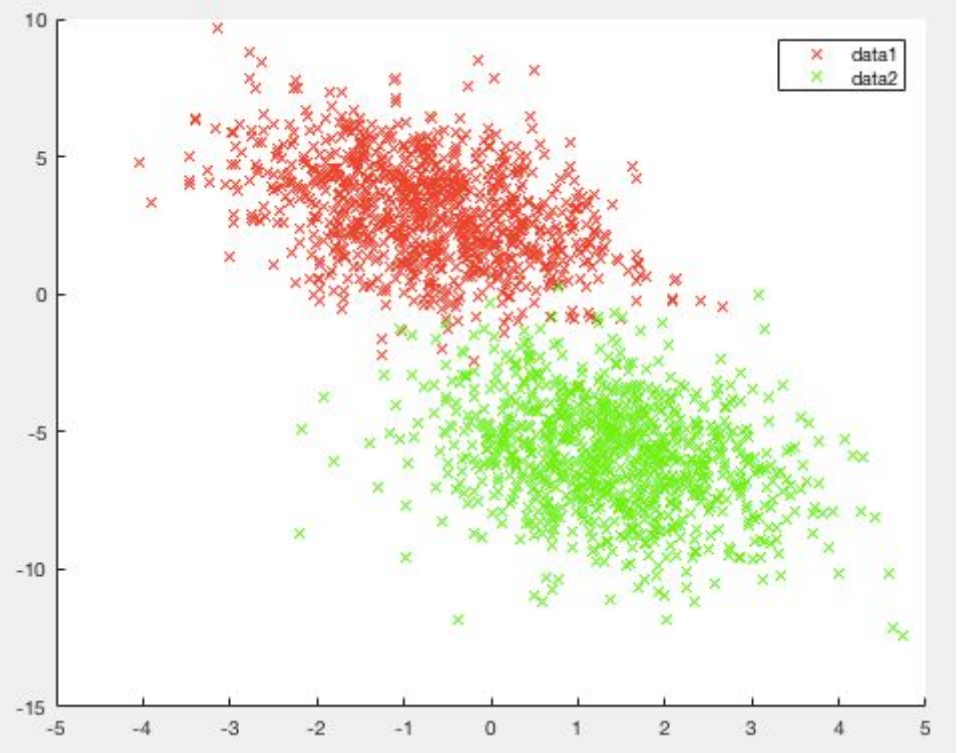
A report by

Adam Nyberg Rasmus Johns
Adany869 Rasjo813

# Data overview and analysis

## Data set 1

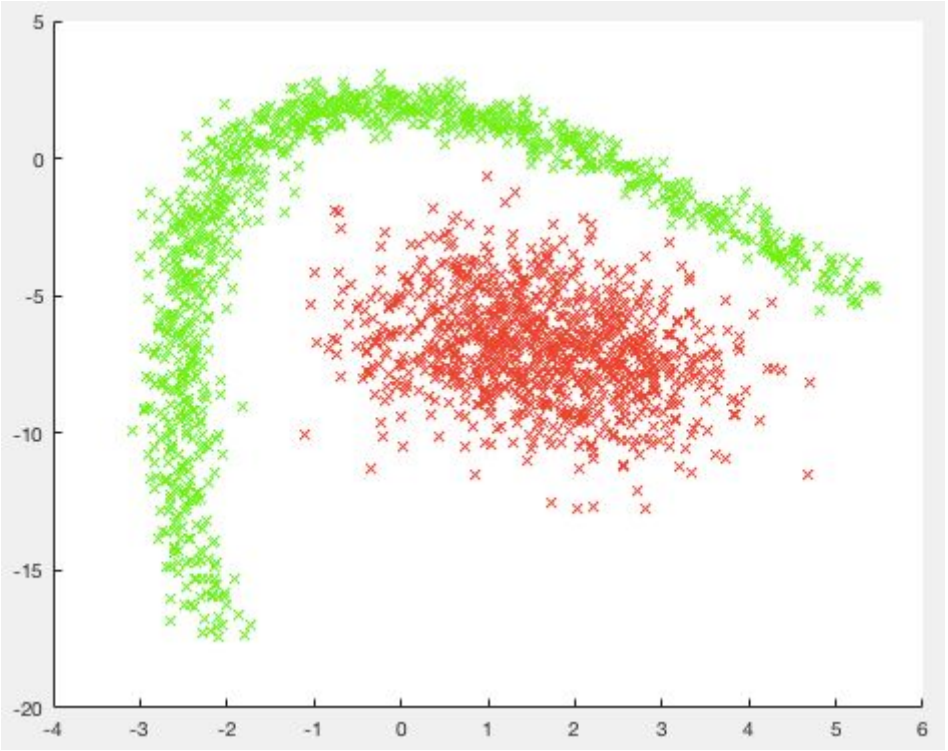
It looks like a linear classifier would be good enough to solve this data set, yet not completely. The two labels are clearly intertwined so if we wanted 100% accuracy our classifier would be very overfitted.




Data set 1

## Data set 2

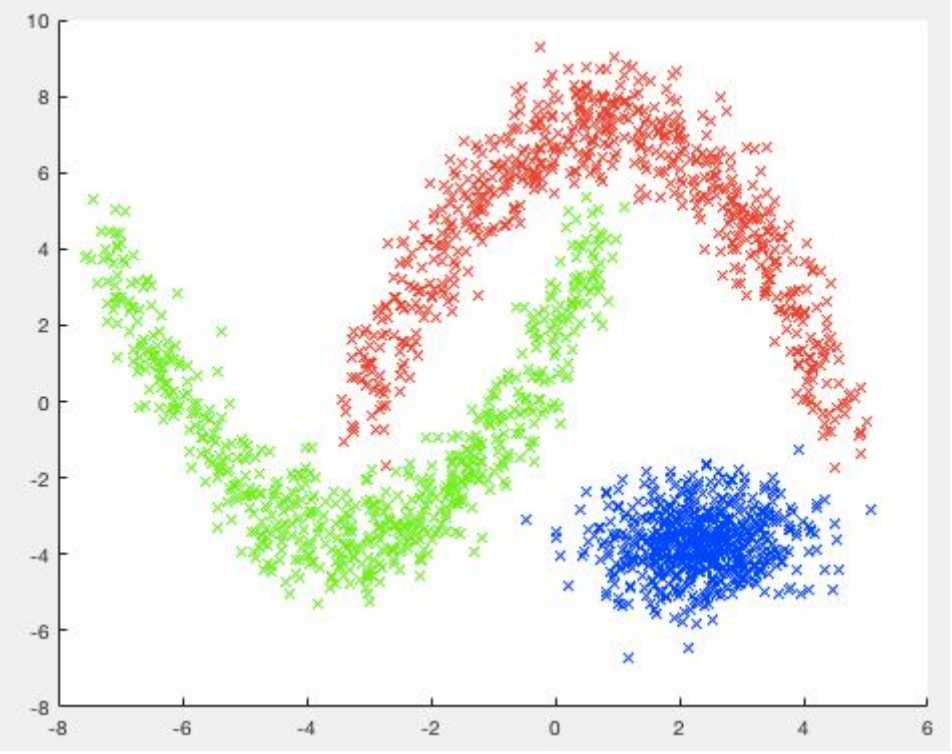
This data set have two clearly separable clusters. They are not separable with a linear classifier so a nonlinear classifier is needed.




Data set 2

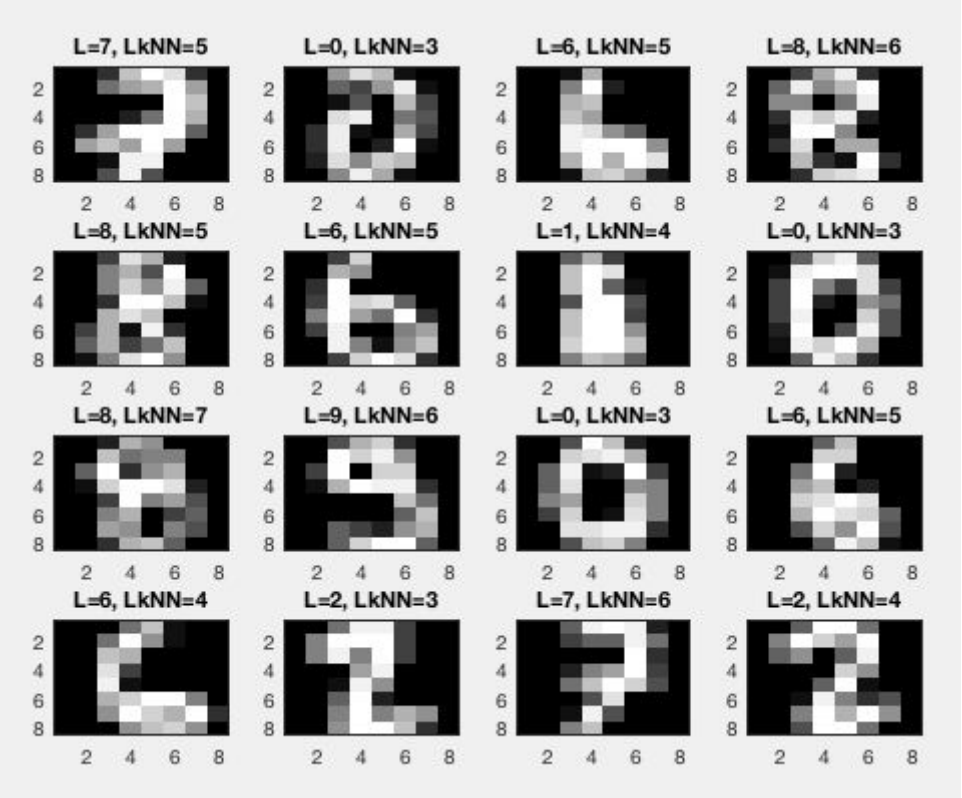
## Data set 3

Data set 3 is clearly divided into three clusters, but with some data points intertwined. The data is not separable with a linear classifier; therefore, a nonlinear classifier is needed to separate the data.



## Data set 4

Data set 4 is the most complex data set. The data set includes the color of all 64 pixel blocks for every training sample. The data looks hard to separate with a linear classifier.




Data set #4

# Necessity of pre-processing

It's more robust in the sense that the down sampling will reduce noise. By down sampling the complexity of the model can be simpler due to the smaller amount of features. Training is also faster when dealing with fewer features.

# Implementation of kNN

First, we calculated our distance matrix using Euclidean distance. Then, iterating from row to row in the distance matrix, we sorted each row in the matrix -- effectively finding the nearest neighbours to each data point. Then, in order to predict the class of our data point in question, we simply looked up the class of the nearest k data point in the sorted distance matrix's row. These neighbours had classes labeled as integer

# Draws in kNN

Seeing how we just sum up class names, which are integers, and find the average value of these neighbours, we handle draws by always rounding upwards. For instance, two data points of class 1 and two data points of class two would mean *(1+1+2+2)/4 = 1.5,* which gets rounded to class 2.

This keeps the predictions simple. Furthermore, ties can always be solved by selecting an odd number of neighbours. In cases where we want an even amount of neighbours, our solution is decent. If one wanted to improve our solution, one could for instance weigh in the distance from the k neighbour to our data point in question. Such a solution would be more f

# Using cross-validation to find best k

In order to find an optimal K, we used cross-validation. We tried k values between 1 and 15.

In our cross-validation, we first divided our data into five folds. By then utilizing cross-validation, meaning we used the holdout-method with each fold as test once, we found a best k value.

# Backpropagation in neural networks

## Single layer

First, the data is sent through the single layered neural network, producing a result. This result is what is called the network's prediction. The prediction can be compared to the correct class label for the data sent through the network. Using this difference between prediction and correct class label, the network can then be trained by backpropagation, meaning the networks coefficients are updated in the opposite direction of the gradient, which is calculated using the derivatives.

## Multi layer

The multi layer network uses the same principle with the difference that it has a single hidden layer. That means that during the training coefficients both before the output and before the hidden layer have to be trained. First we calculate the gradient of the coefficients before the output layer using the following formula.

$$V = V - learningRate * (\frac{2}{N} * ((W^T(Z - Y)). * \sigma'(S))X^T$$

We also use a similar formula for the coefficients before the hidden layer. We run these calculations once every iteration and the idea is that for every iteration the coefficients should be a little better.

**Neural network result** (• Present the results form the backprop training and how you reached the accuracy criteria for each dataset. Motivate your choice of network for each dataset, then 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. )

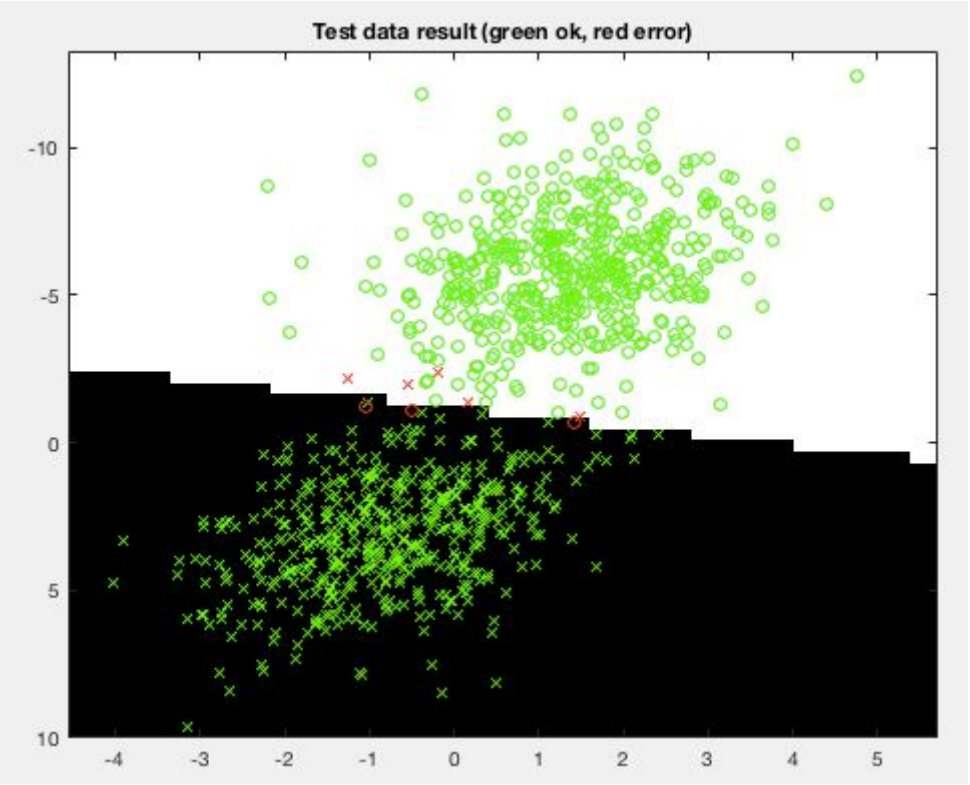
The result from the backprop training is presented in the table below. When deciding on number of iterations we just kept increasing it until the error converged. For the learning rate we tried to have it as high as possible so that we didn't have to have too many iterations.

| Dataset | # Hidden neurons | # Iterations | Learning rate | Accuracy |
|---|---|---|---|---|
| 1 | 2 | 15000 | 0.003 | 0.99 |
| 2 | 13 | 10000 | 0.03 | 1 |
| 3 | 13 | 10000 | 0.03 | 0.998 |
| 4 | 130 | 10000 | 0.021 | 0.96282 |

## Data set 1

We wanted to classify this dataset using few neurons, seeing how the data could be well separated by a single straight line. Therefore, we started experimented using only one neuron but was not able to achieve the required accuracy. We then tried using two neurons and settled on the following parameters:

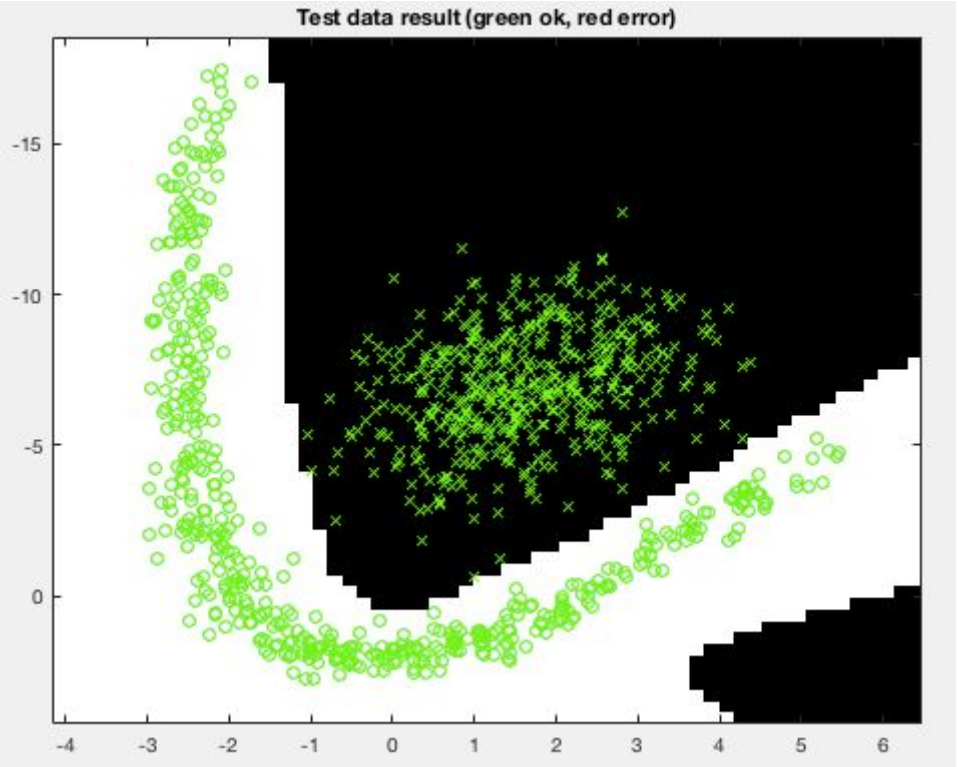
| Dataset | # Hidden neurons | # Iterations | Learning rate | Accuracy |
|---|---|---|---|---|
| 1 | 2 | 15000 | 0.003 | 0.99 |




## Data set 2

This dataset was clearly separable, but seemed to requier a higher degree of neurons in order to find a good curve. We therefore raised our number of neurons and raised our learning rate from the previous example.

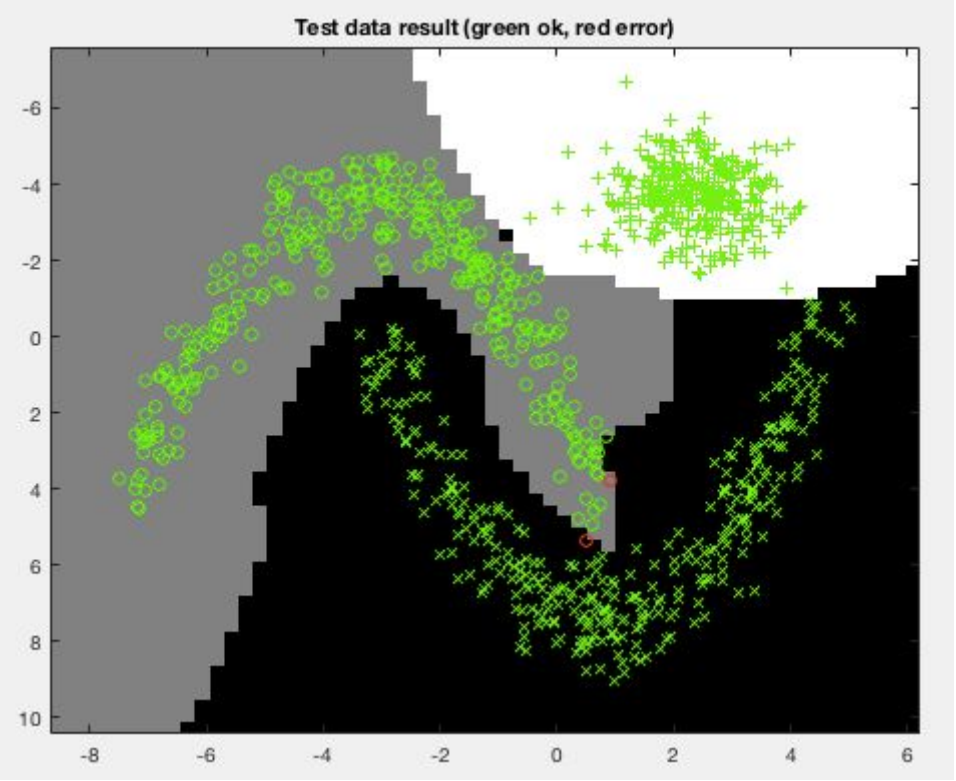
| Dataset | # Hidden neurons | # Iterations | Learning rate | Accuracy |
|---|---|---|---|---|
| 2 | 13 | 10000 | 0.03 | 1 |




## Data set 3

This dataset resembled Data set 2 quite a lot -- the major difference was the introduction of another class. We therefore did something which worked out great: we just used the same amount of neurons and training as in Data set 2 which turned out to work really good on this data set as well.

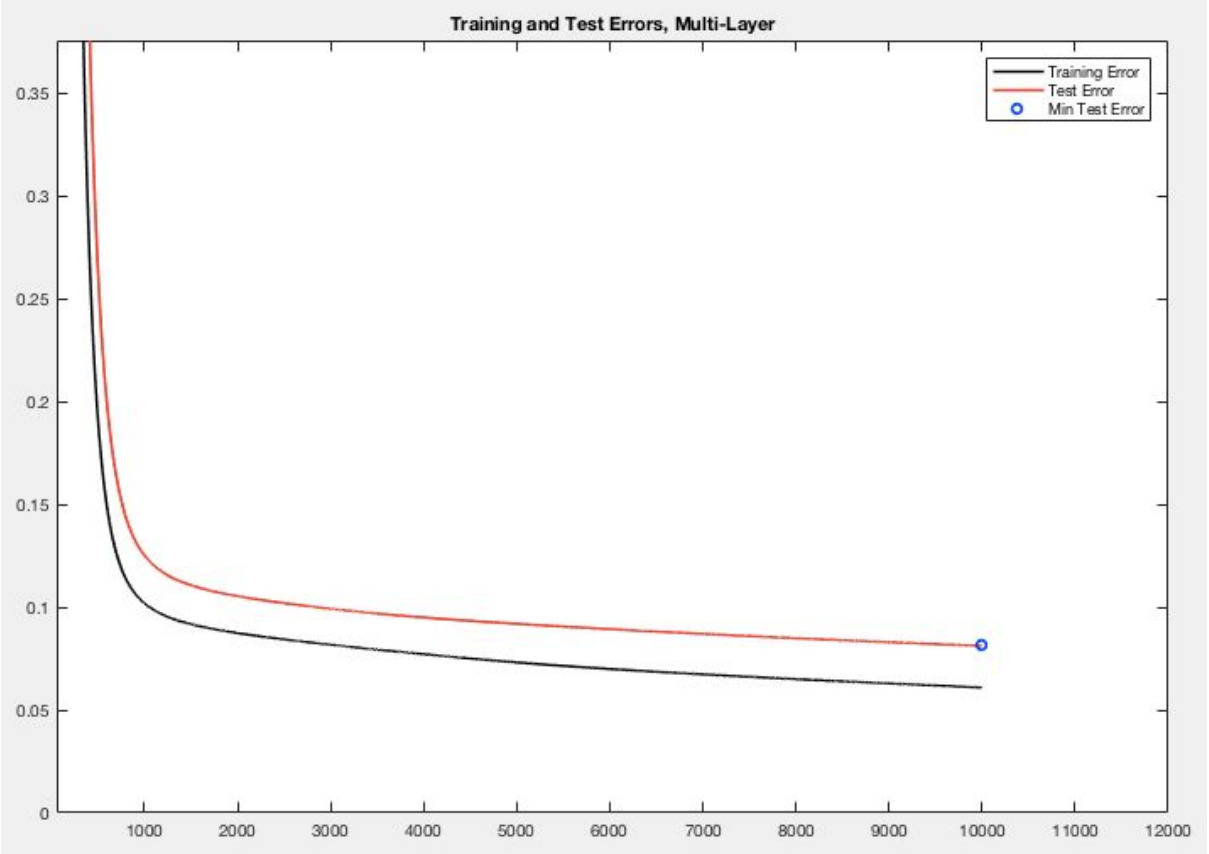
| Dataset | # Hidden neurons | # Iterations | Learning rate | Accuracy |
|---|---|---|---|---|
| 3 | 13 | 10000 | 0.03 | 0.998 |




## Data set 4

This dataset was the most difficult to train a neural network for. First, we attempted to train our network as with previous data set: using qualified guesses for parameters. However, this proved unsuccessful. Therefore, we decided to normalize our data between -1 and 1 for this task. Then, we could clearly see the network getting trained a lot better than previously; from that point, it was easy to simply try different amount of neurons and learning rates until we found a good match.

| Dataset | # Hidden neurons | # Iterations | Learning rate | Accuracy |
|---|---|---|---|---|
| 4 | 130 | 10000 | 0.021 | 0.96282 |

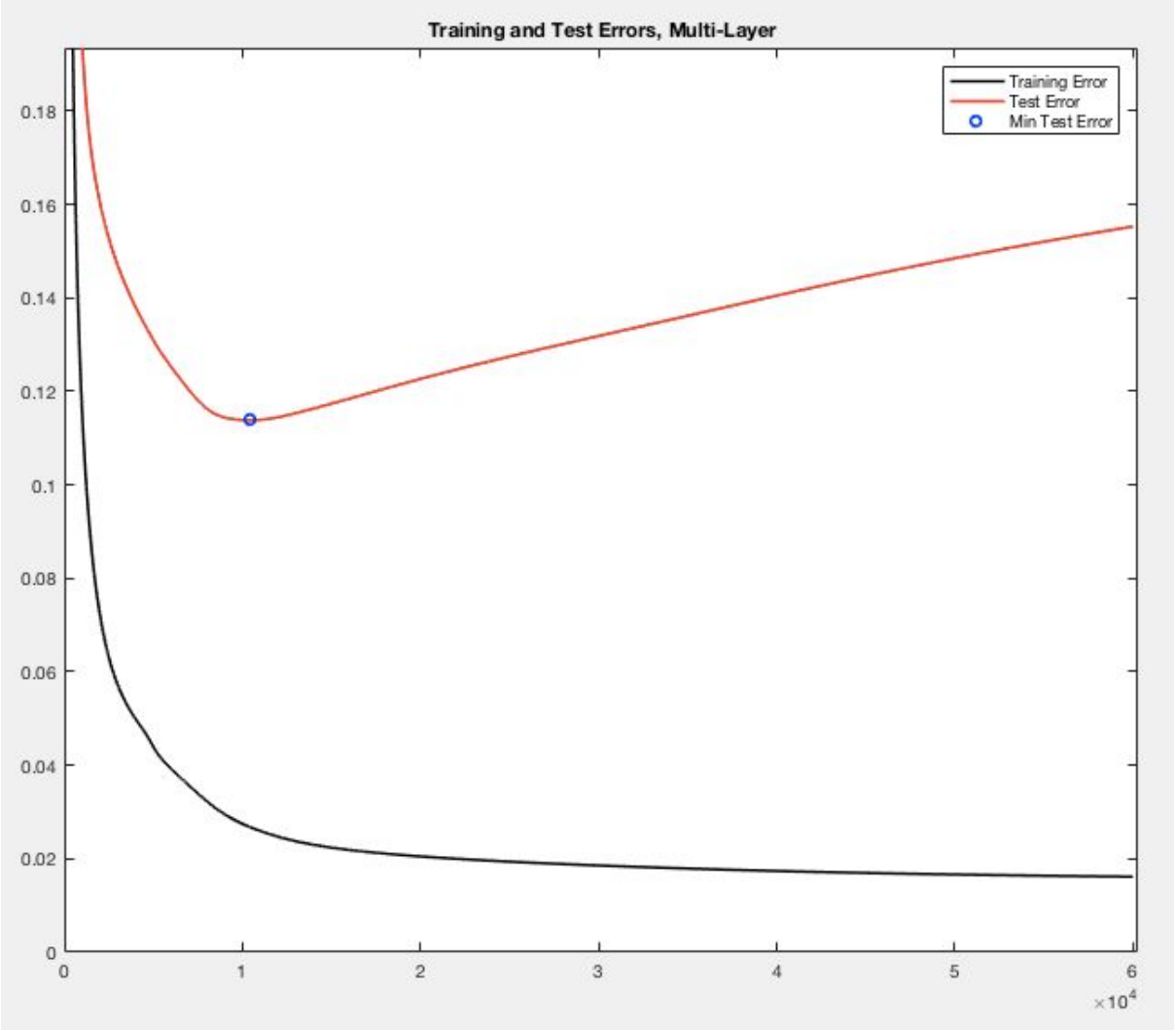



## Non-generalisable backpropagation result

We first divided the data from dataset 3 into 50 bins, using one of them for training and another one for test. We then trained the neural network with the following parameters:

| Dataset | # Hidden neurons | # Iterations | Learning rate | Accuracy on test |
|---|---|---|---|---|
| 3 | 13 | 60000 | 0.01 | 0.97436 |

However, since we divided the data into very small sets, the risk of them varying from one and another increased tremendously. That way our network will be very overfitted and not able to generalize. Looking at the result below, the result of this phenomenon is illustrated:




## Discussion

KNN is in its essence very simple and intuitive. One can easily understand how the classification is done.

KNN has the potential to classify data *very* well given a good distance measurement and sufficient data.

KNN can be computationally expensive if there is a huge data set to base predictions on.

KNN requires some sort of distance measurement, which can be tough to decide. For instance, the data might have many dimensions and and vary a lot in some dimensions, while being compact in other dimensions. Simply using an Euclidian distance in such a case would make the algorithm almost entirely base its predictions on the dimensions that vary the most, seeing how they have the potential to impact the distance.

A huge advantage of neural network is that it does not have to save the data once it has been trained. Simply saving the networks parameters is enough.

Neural networks works good on highly dimensional and non linear data.

A draw back of neural networks is that it takes long time to train and can be time consuming to pick the right parameters and architecture. Another draw back is that neural networks requires a lot of training data.