TBMI26 – Computer Assignment Report  
Supervised Learning

Deadline – March 15 2019

Author/-s:

Isak Johansson-Åkhe, Omid Tern

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. **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 data from a machine learning perspective. Consider if you need linear or non-linear classifiers etc.**

Dataset 1: Two clearly separated gaussian distributions, with little overlap. Can easily be separated by a line, meaning only a linear classifier is necessary.

Dataset 2: While one set is like before, the other reaches around it in a manner which makes linear separation impossible. This can be solved with a simple transform of the data, meaning a non-linear classifier is necessary, albeit not a complicated one.

Dataset 3: Much like in the last case, the different sets envelop each other. This time there is also the complication of having three different distributions. Same verdict as last dataset.

Dataset 4: Very complicated dataset with 65 features requiring complicated conclusions to be drawn as to how they interact with each other to form numbers. Absolutely necessary to use a non-linear classifier. Doubt it is even possible to train a linear one to any effect.

1. **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**](http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits)

Averaging over pixels this way drowns out noise in the data, hiding smaller faults and distortions.

1. **Give a short summery of how you implemented the kNN algorithm.**

For each datapoint to be evaluated, the Euclidian distance between that point and each datapoint in the training set is calculated. Next, the list of training-data labels is sorted according to these distances, and the labels of the first k samples are counted. The label which occurs most times is assigned to the evaluated datapoint.

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

In the case that the label with most votes and the one with second-most votes have the same amount of votes, the list of training-data labels as sorted by the distances are iterated over until a label with as many votes as the top label is encountered, and it is this label which is assigned to the evaluated datapoint. In summary, if several labels get equally many votes, the classification is determined by which of the points with these labels is closest to the evaluated datapoint.

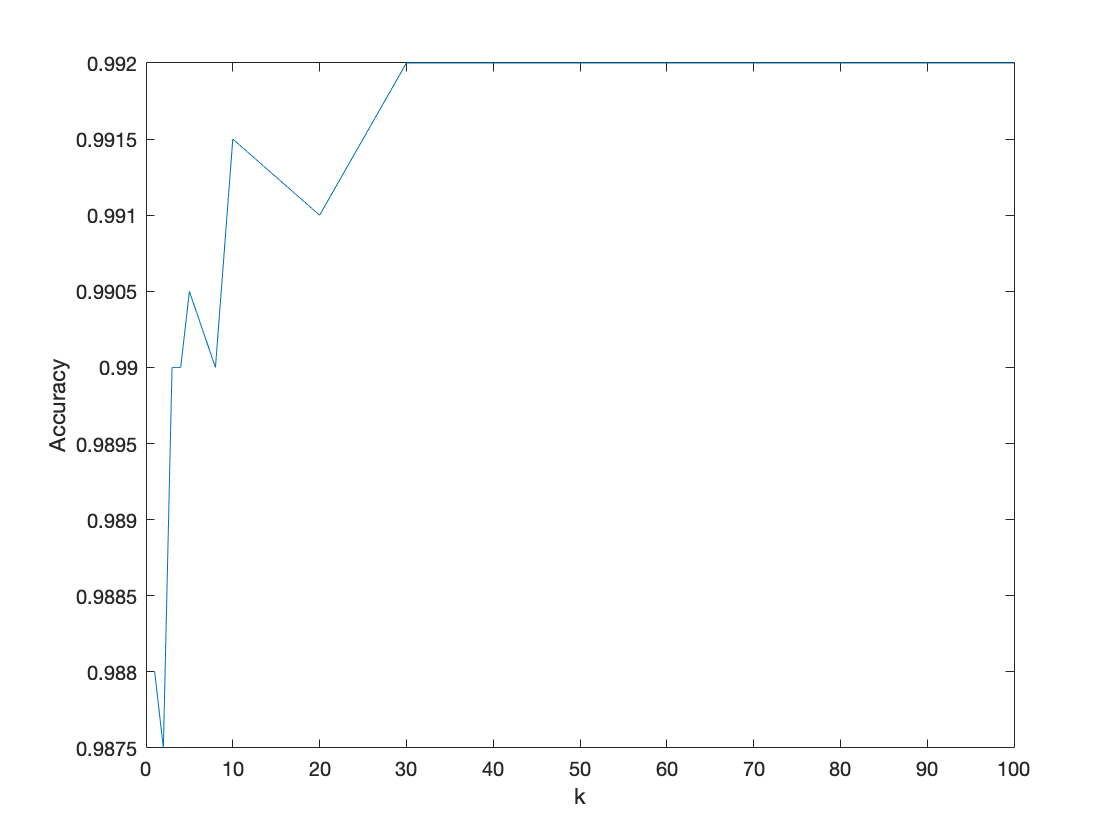
1. **Explain how you selected the best k for each dataset using cross validation. Include the accuracy and images of your results for each dataset.**

Cross-validation was implemented at the end of the “evaluate\_kNN.m” script, with the help of a function “cross\_validate\_kNN.m”.

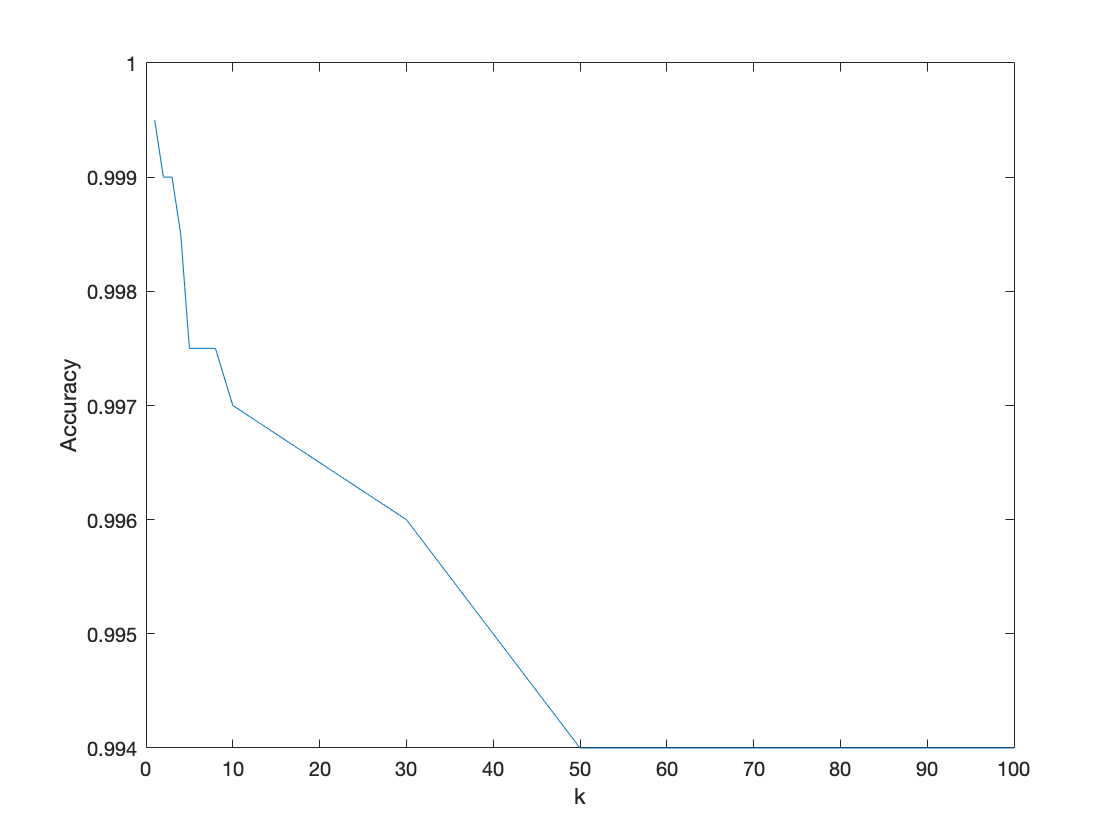
For each dataset, all the datapoints were evaluated by the algorithm by 8-fold cross-validation, for different k ranging from 1 to 100. The cross-validation was done by randomly dividing the dataset into 8 equal bins, and then taking turns using 7 of the bins as training data and 1 as testing data, until all bins have been evaluated. The results are concatenated and have their accuracy evaluated.

For selecting the best k, we produced a k vs. Accuracy graph for each dataset, and picked the k which simultaneously gave a high Accuracy and produced a somewhat stable predictor (which we can tell from how much the Accuracy varies in regards to nearby k).

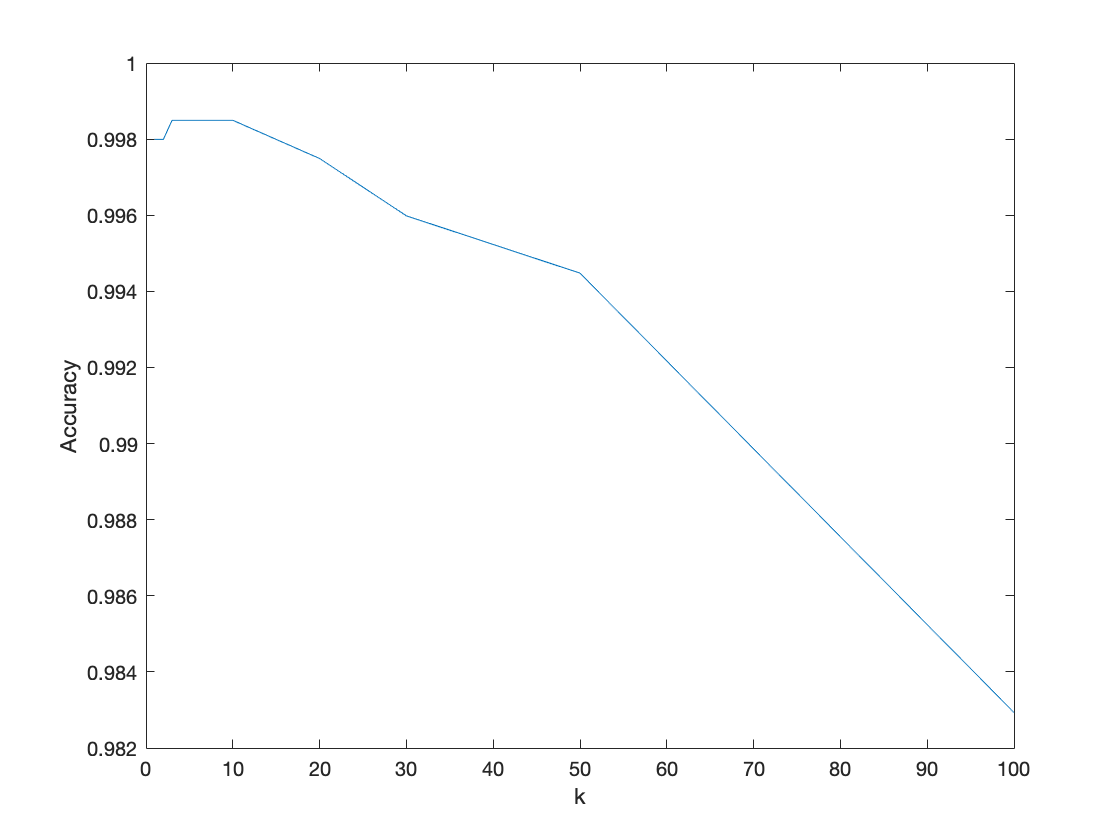
Dataset 1: k = 40, Acc = 99.2%. Setting k as high as possible would be good, but also makes it slower (albeit very little).



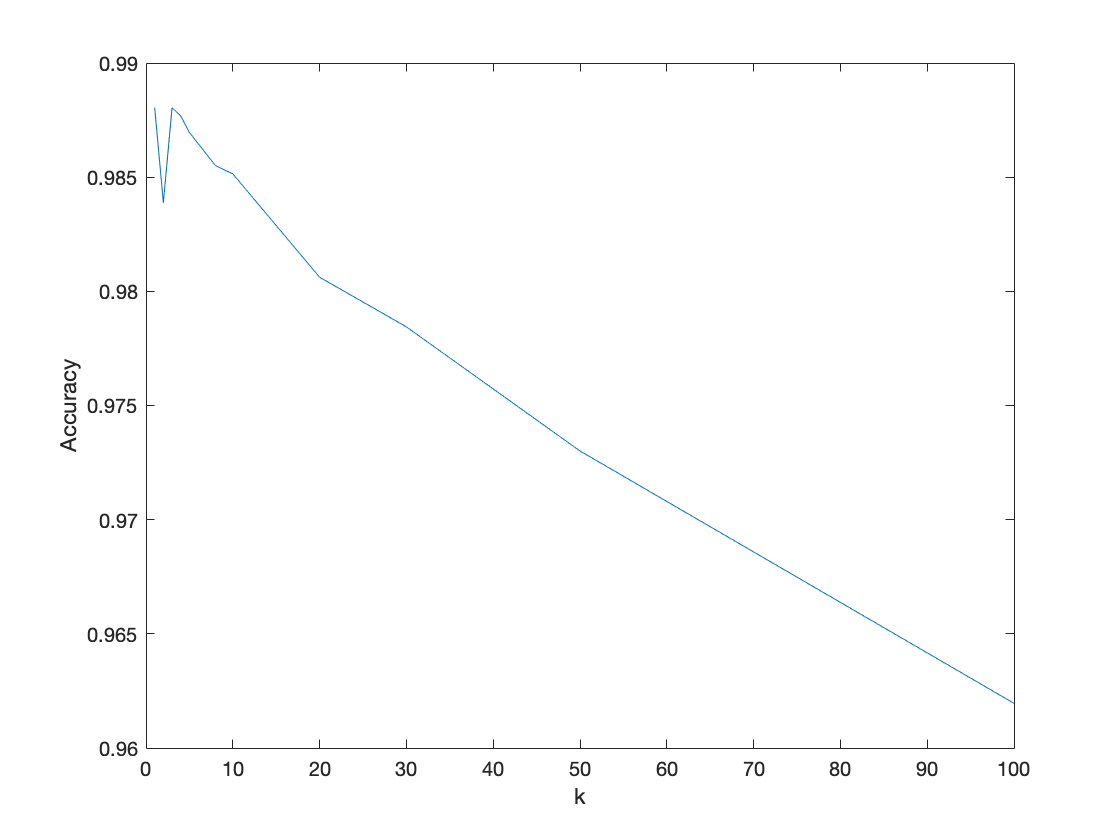
Dataset 2: k = 2, 99.9%. Should set k as low as possible, but would be pretty unstable if set to 1.



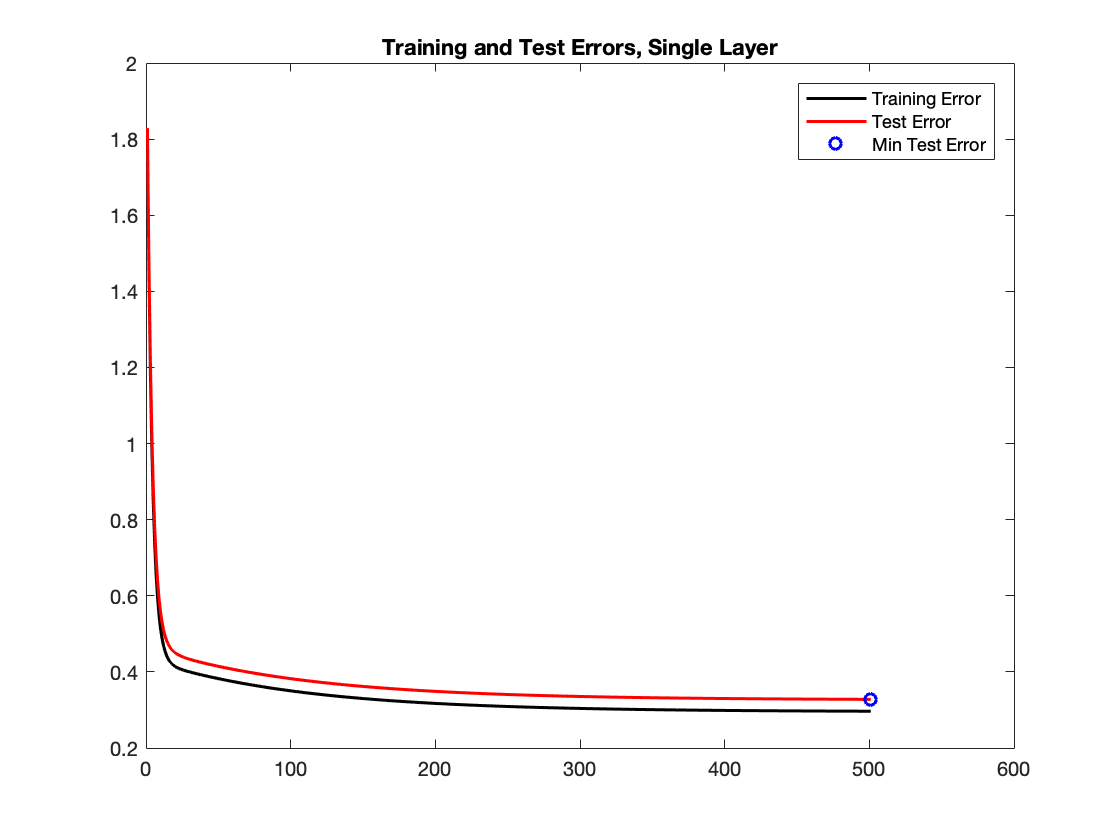
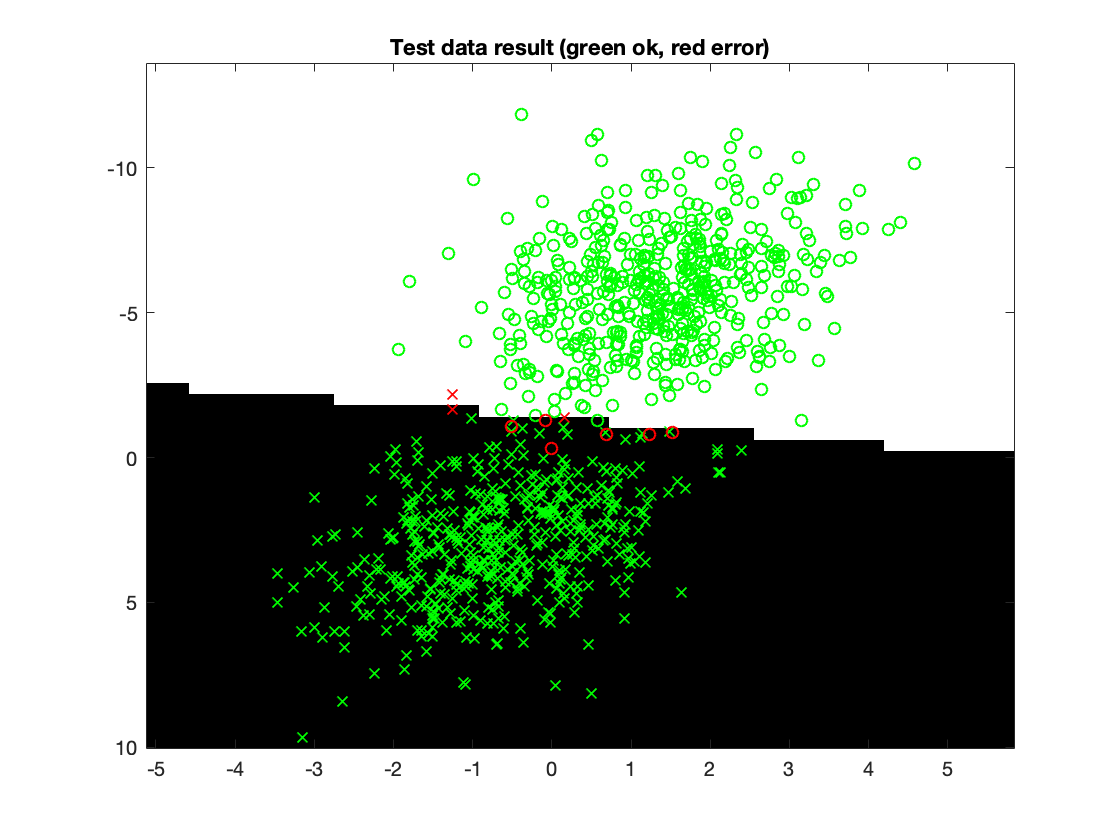
Dataset 3: k = 8, Acc = 99.85%. Seems to get an initial small boost in performance when setting k higher than 2 before performance starts to drop.



Dataset 4: k = 4, Acc = 98.56%. Much like in dataset 2, it should be kept low, but too low seems unstable.

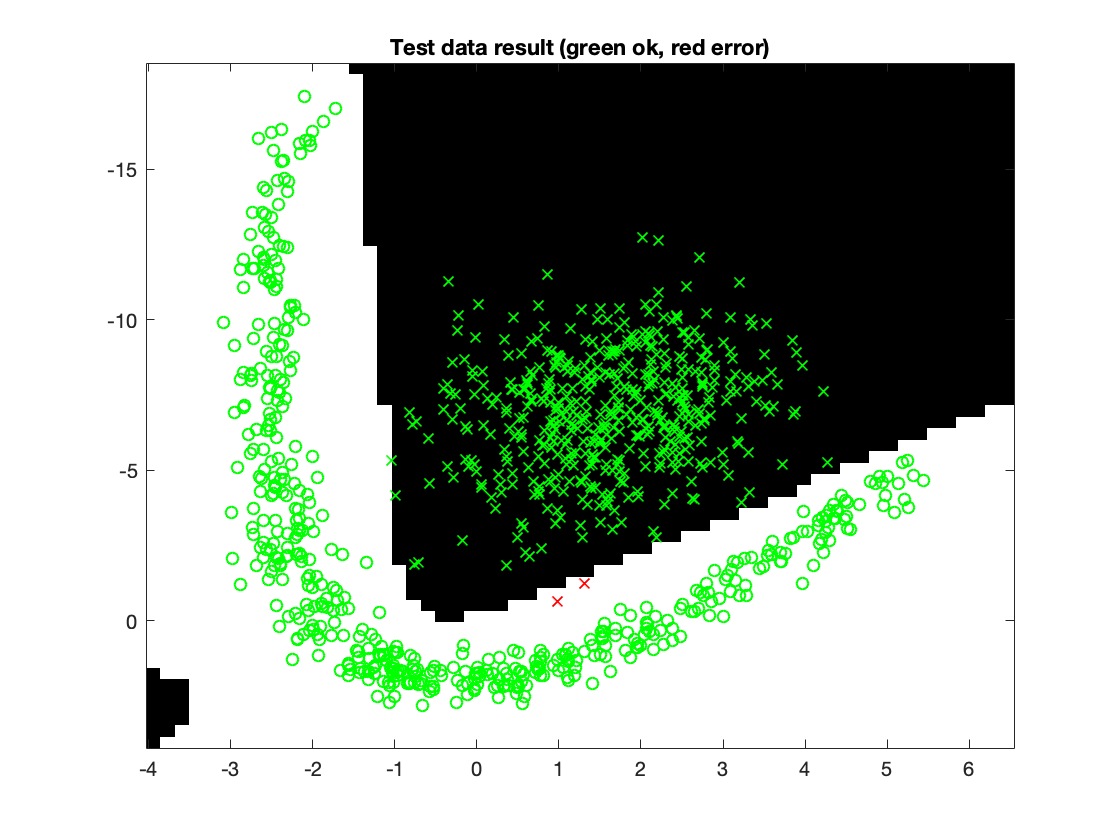
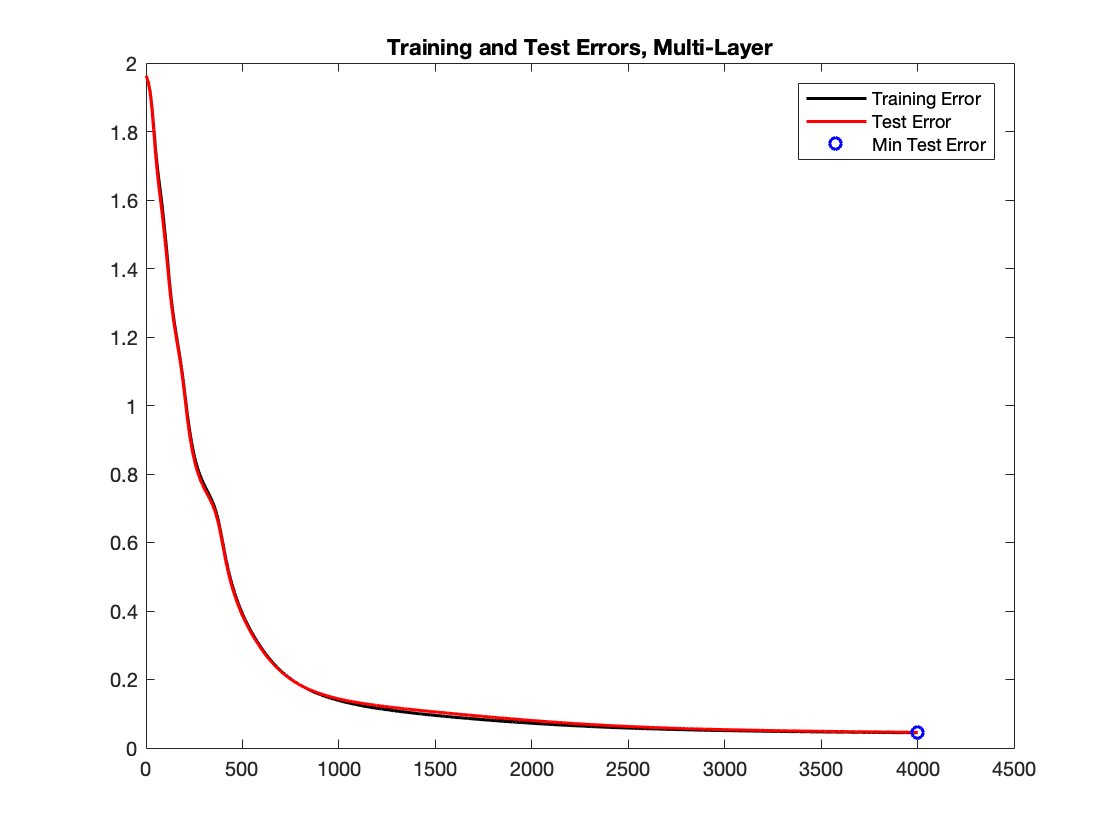


1. **Give a short summery of your backprop network implementations (single + multi). You do not need to derive the update rules.**
2. Initialize weights to random values close to 0
3. Calculate network output by running the network
4. Calculate gradient with help of output (different gradient update functions for different layers when applicable)
5. Update weights via step function (with help of gradient)
6. Calculate errors
7. Repeat 2 through 5 for a given number of epochs
8. **Present the results from the backprop 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.**

****

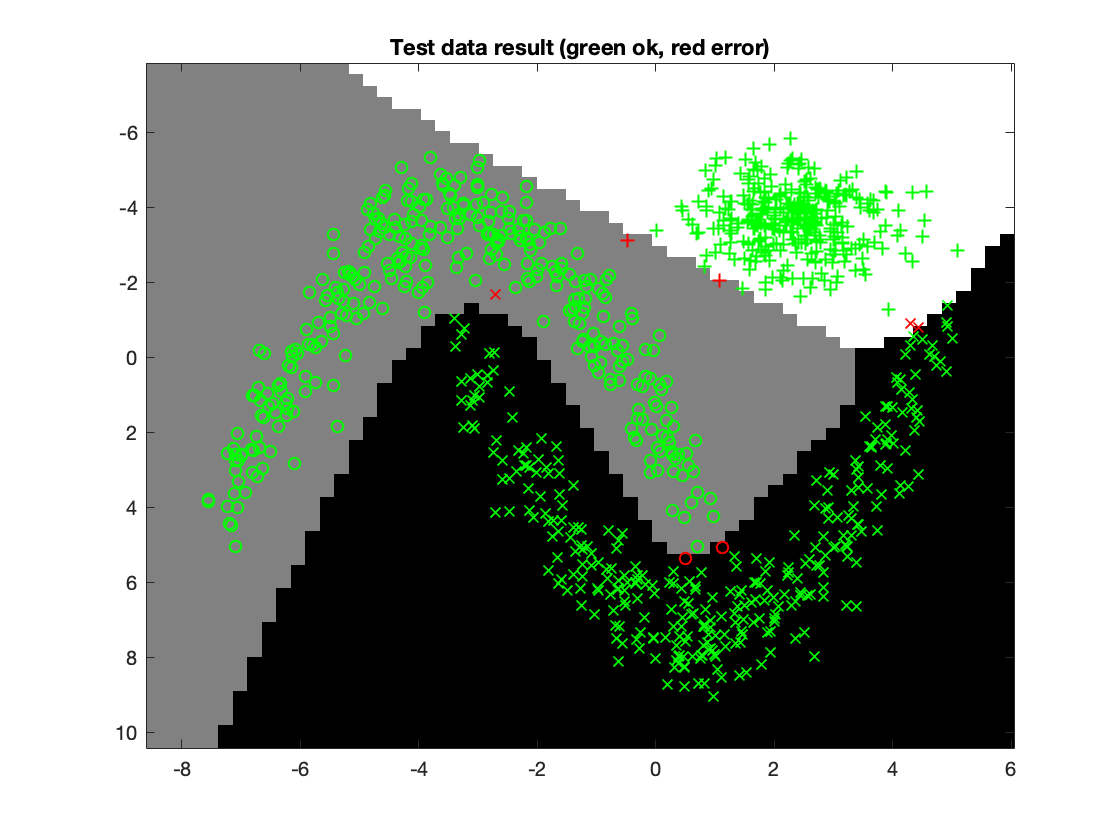
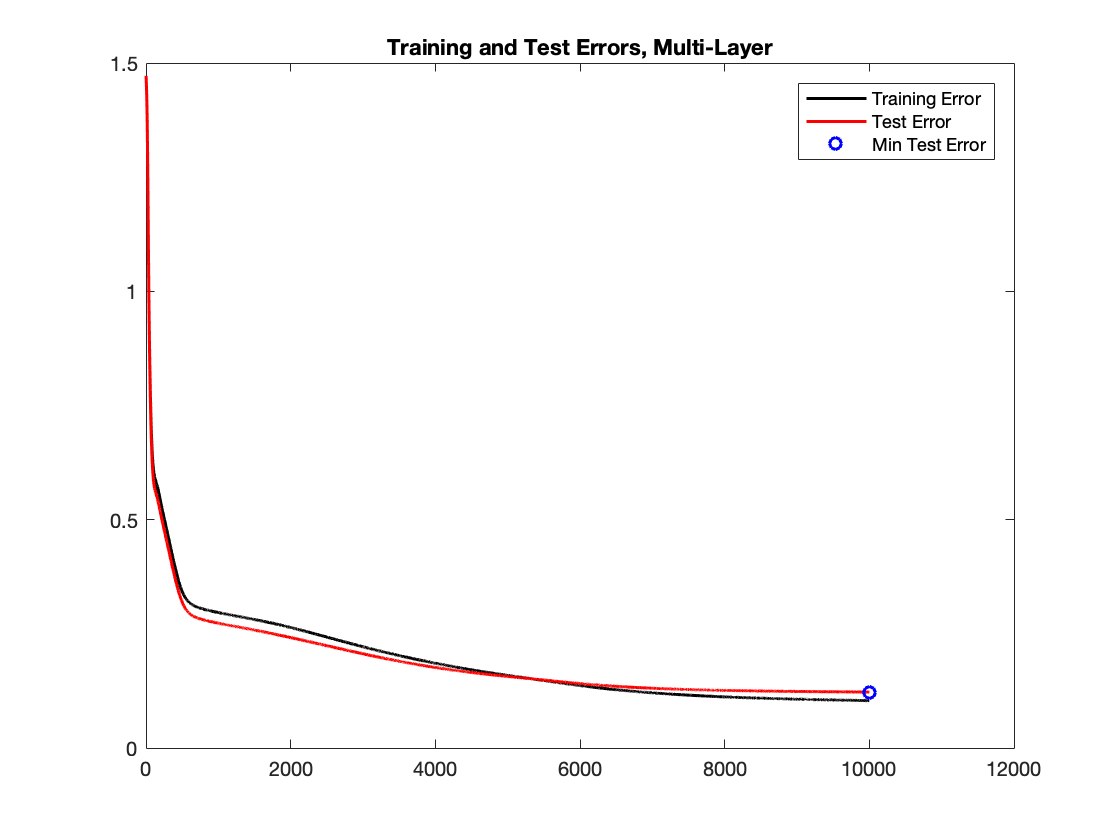
Dataset 1: --Single Layer Network--

* Accuracy: 99.1%
* Training Time: 0.0410 s
* Epochs: 500
* Learning Rate: 0.005
* Hidden Neurons: -
* Motivation: Linearly separable dataset, which means it should have no need for any hidden layers. Additionally, epochs can be set low and learning rate high as the network will converge early.



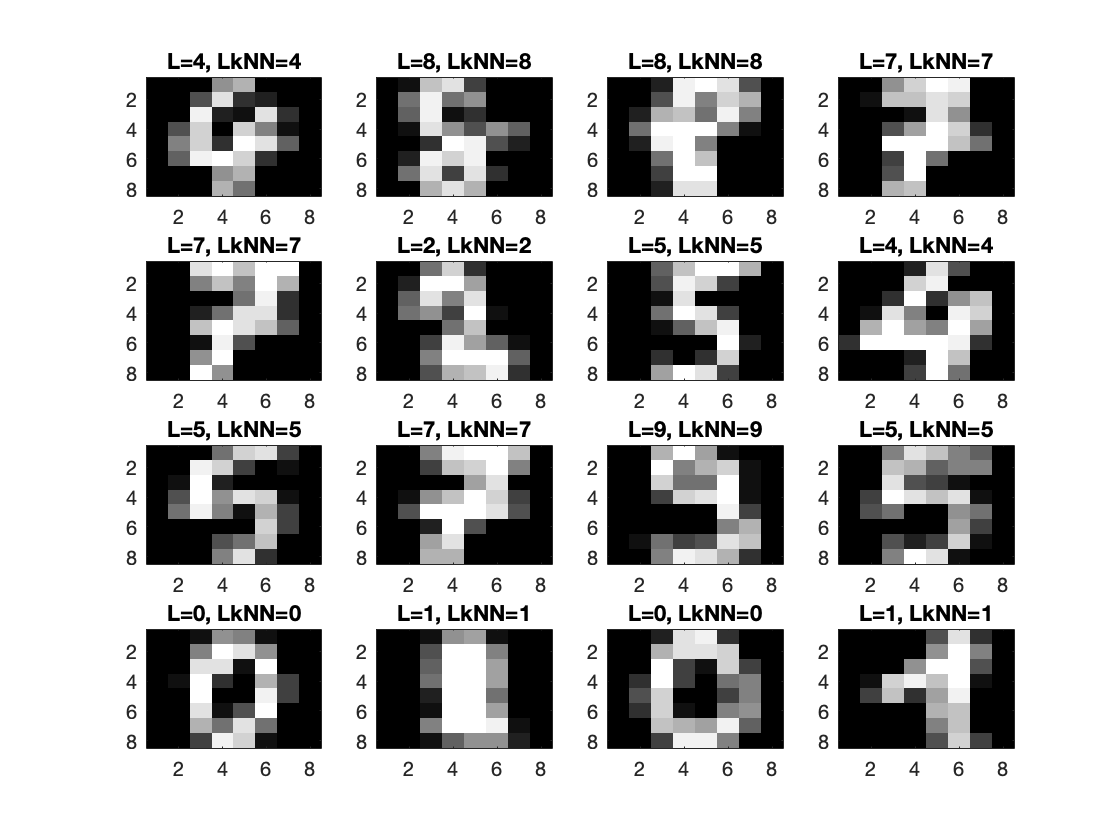
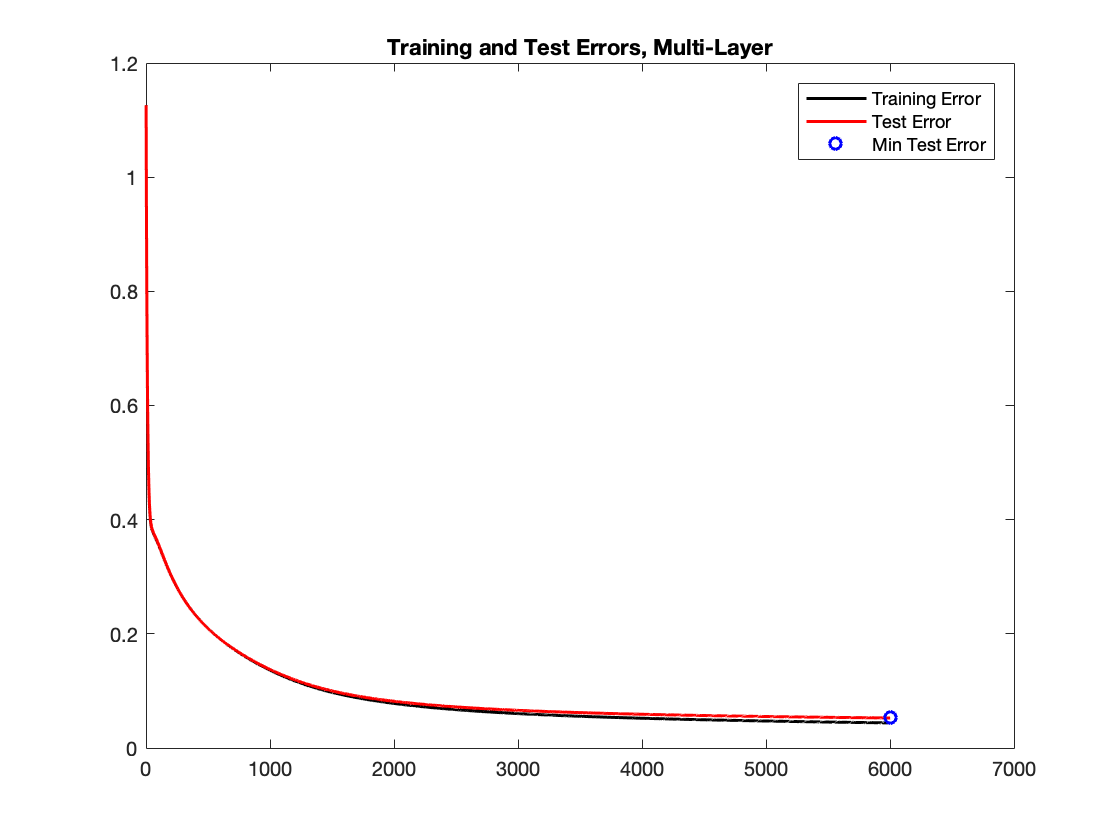
Dataset 2:

* Accuracy: 99.8%
* Training Time: 1.1839 s
* Epochs: 4000
* Learning Rate: 0.01 (Can be high, as there is ample space between classes and even sharp lines will separate the dataset nicely)
* Hidden Neurons: 3 (Minimum amount required to get the bent shape required to fit the data)
* Motivation: The distance between datapoints of different labels is large, so the network can afford to work fast and dirty. Additionally, the data only really needs one more dimension to be linearly separable, so only 3 hidden neurons are needed.



Dataset 3:

* Accuracy: 99.3%
* Training Time: 4.5765 s
* Epochs: 10000
* Learning Rate: 0.01
* Hidden Neurons: 6
* Motivation: Because of the more complicated nature of the separation of data, as well as the increase in number of neurons (which in turn also is because of the complexity), the network needed more epochs to train. The number of hidden neurons is the lowest number that can describe a separation like that of the image with relative consistency, keeping training time low by not adding unnecessary neurons to train. Clear separations between labels and few outliers means we can still have a fast learning rate.

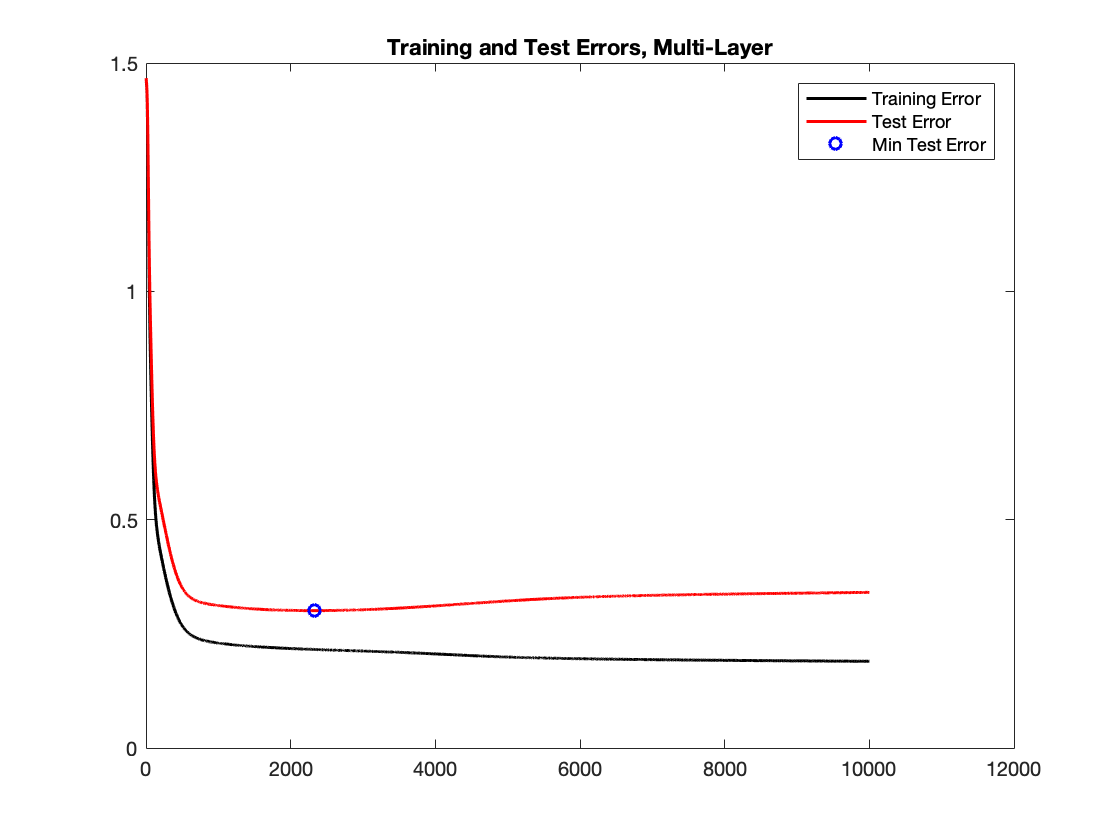


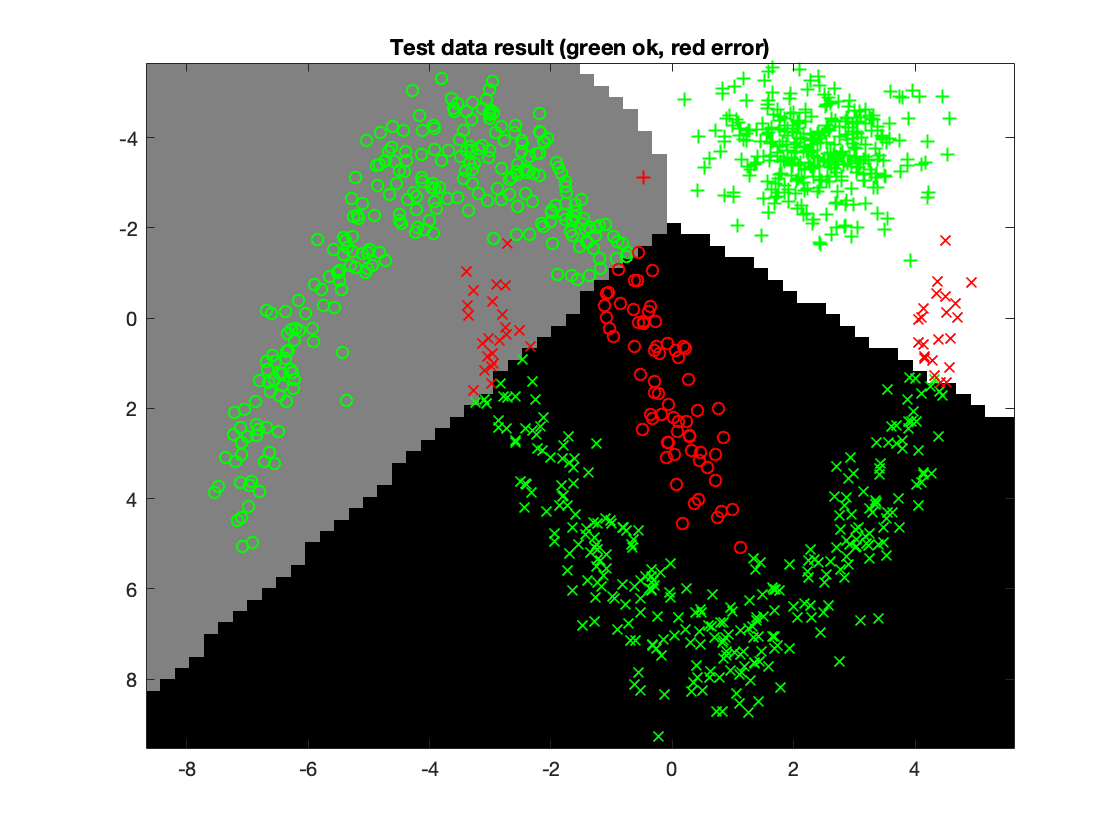
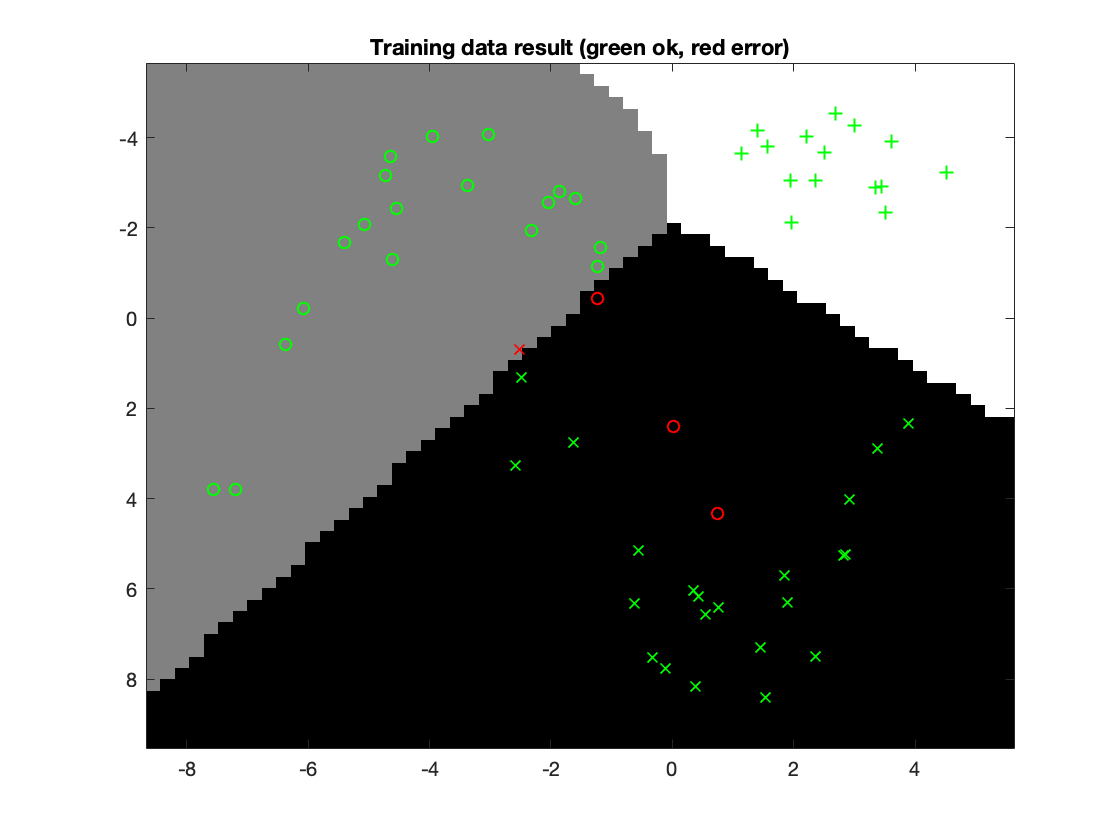
Dataset 4:

* Accuracy: 96.5%
* Training Time: 62.6811 s
* Epochs: 6000
* Learning Rate: 0.001
* Hidden Neurons: 33
* Motivation: This dataset had a much higher number of relevant features, and 9 output labels. As such, many more hidden neurons were required. Additionally, the learning rate had to be decreased, probably because of the close similarity between many targets, leading to an unstable predictor if the learning rate is just a little too high.

1. **Present the results, including images, of your example of a non-generalizable backprop solution. Explain why this example is non-generalizable.**

Changes compared to optimized solution: Used only 60 datapoints for training. Used only 3 neurons for the hidden layer.

****

****

The low number of training points causes the network to be able to make a passable fit to the training points in a manner which does not represent the distributions they hail from. The few points at the ends of the crescent shapes for instance can be seen as outliers. Additionally, with fewer neurons in the hidden layer, it becomes impossible to describe the complicated borders between label sets.

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

For these datasets, kNN works particularly well. There is very little noise and outliers, which suits kNN well, meaning we can often keep k low. However, for sets like set 4, there is a lot of training points available which means each prediction will take a lot of time.

Compared with kNN, neural networks still perform well, albeit with somewhat lesser performance for the more complicated sets, as it needs to find a complex combination of weights in a possibly hard to reach global minima. Training takes a lot more time, but testing is extremely quick. For set 4 for instance, if it were a real-world application where the classifier would be queried multiple times, the slow nature of the kNN would quickly become annoying, whereas the network would answer almost instantaneously.

It would be interesting to see the classifiers compared against a dataset with more outliers.

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

It might be interesting to try different activation-functions for the neural network. For instance, a gaussian activation function might very easily identify the upper-right distribution in dataset 3, and might converge to an excellently generalizable solution for dataset 2 in much shorter time.

Additionally, for dataset 2 the problem could become linearly separable by converting to polar coordinates, and this might simplify the solution of set 3 as well.