TBMI26 – Computer Assignment Report

Supervised Learning

Authors:

Ludvig Knast ludkn080, Erik Jareman erija971

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: Two classes clearly separated, linear classifier is enough.

Dataset 2: Two clearly separated classes but cannot be separated by straight line, therefore a non-linear classifier is needed.

Dataset 3: Three clearly separated classes that cannot be separated by straight line, therefore a non-linear classifier is needed.

Dataset 4: Multiple classes, complex feature space: a non-linear classifier is needed.

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**

By reducing the dimensionality and noise it therefore also reduces the complexity which eases the data classification and also the training time.

3. Give a short summary of how you implemented the kNN algorithm.

For each sample:

- Calculate euclidean distance to each training sample.
- Sort euclidean distances.
- Select k-nearest neighbors.
- Majority vote of selected neighbor classes decides the predicted label.

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

Predict as first label encountered.

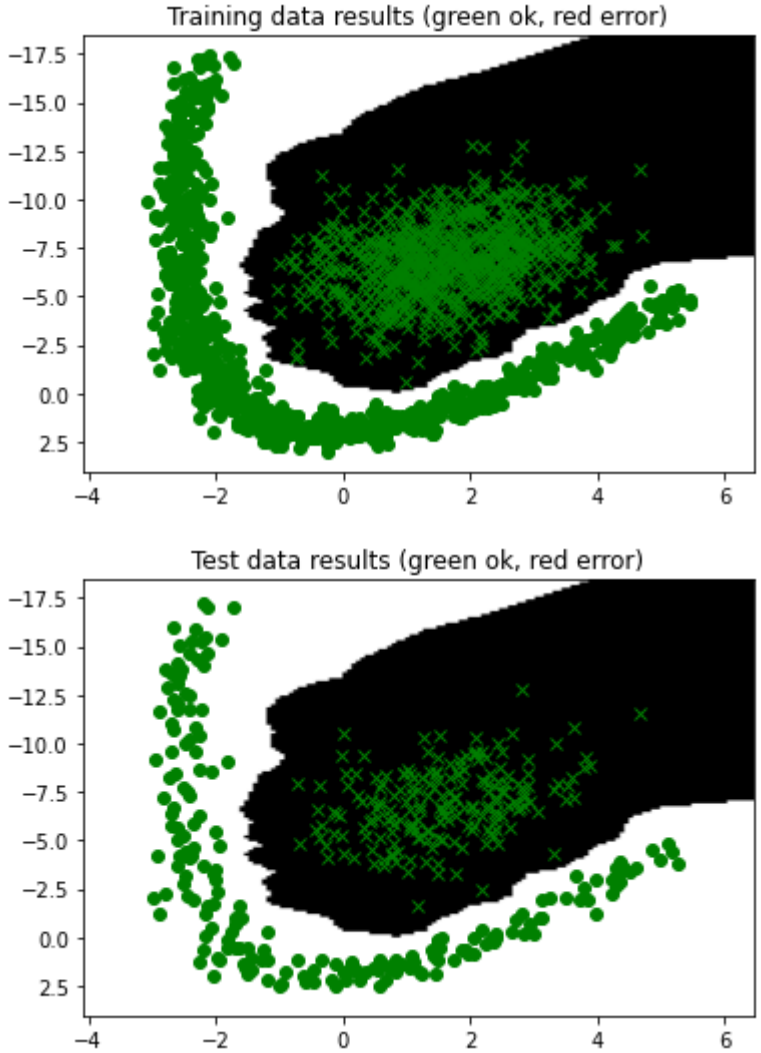
5. Explain how you selected the best k for each dataset using cross validation. Include the accuracy and images of your results for each dataset.

Cross Validation:

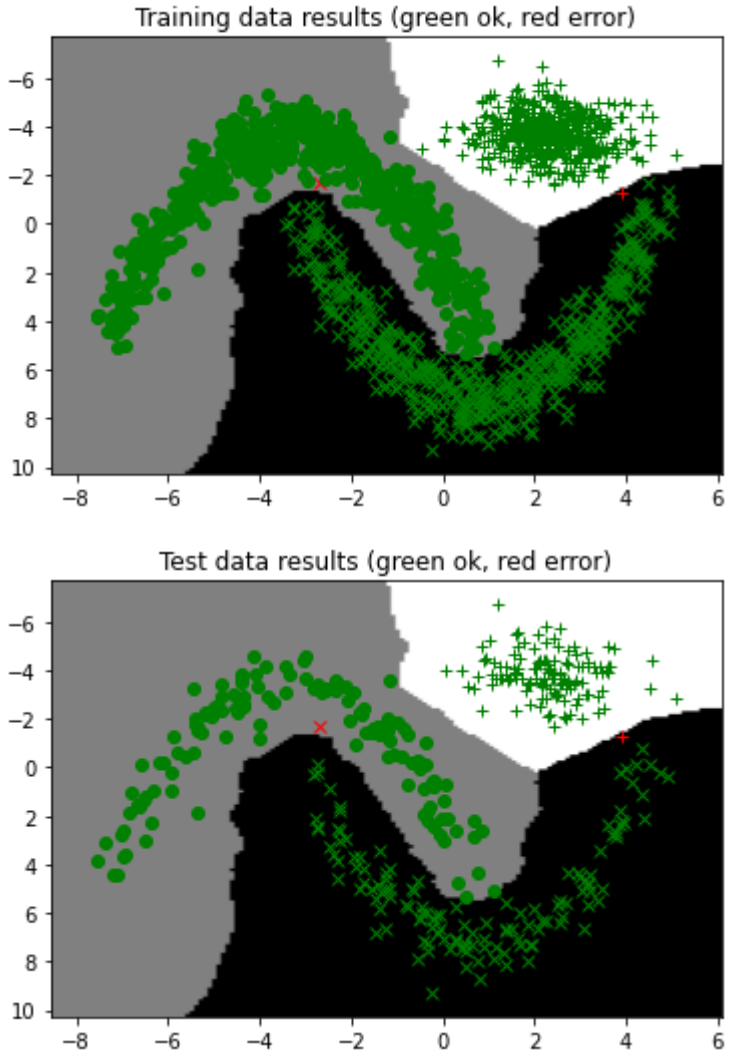
We use cross validation on numBins-1 bins, where one bin at a time is used as a validation-bin. The other bins are used to train the model for different k-values. The accuracy when the trained model is used to classify the validation-bin samples is then recorded. This process is then repeated until each bin has acted as validation-bin for each possible k-value, and the k-value with the highest accumulated accuracy for all of the validation-bins is then chosen as best k.

Dataset: 1. Optimal k: 18. Accuracy (test dataset): 100%.

Dataset: 2. Optimal k: 1. Accuracy (test dataset): 100%.



Dataset: 3. Optimal k: 3. Accuracy (test dataset): 99.5%.



Dataset: 4. Optimal k: 3. Accuracy (test dataset): 99.0%.

[‘no_image’]

6. Give a short summary of your backprop implementations (single + multi). You do not need to derive the update rules.

Single layer:

- Calculate the gradient based on the difference between predicted output values and the desired output values.
- Take a learning step (update the weight matrix).

Multi layer:

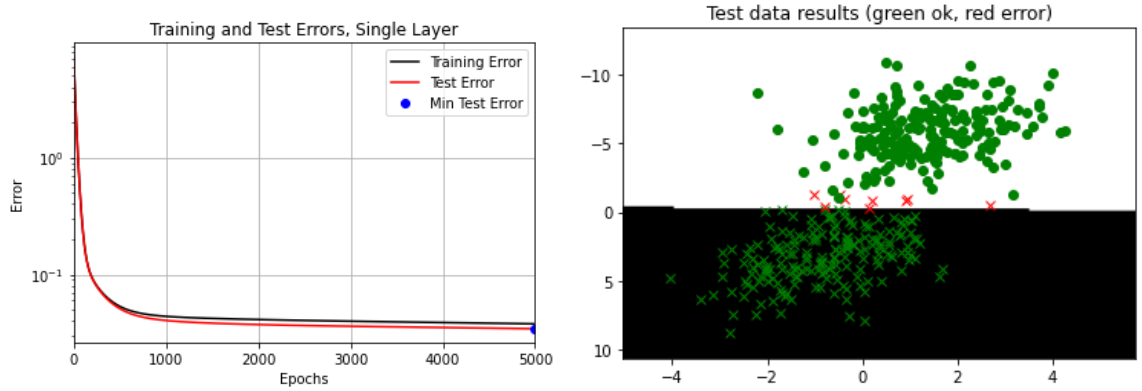
- Calculate the gradient of the last weight matrix based on predicted output values and the desired output values (output layer).
- Multiply the derivative of the activation function in the hidden layer with the error in respect to the weights V and the output error.
- Take a learning step (update both weight matrices).

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:

Low number of both inputs and outputs → low number of hidden neurons (**5**).
Number of iterations: **5000** seems to give it enough time to learn whilst not overfitting.
Learning rate: **0.01** gives a smooth learning curve for this dataset.

Accuracy: 99.75%.

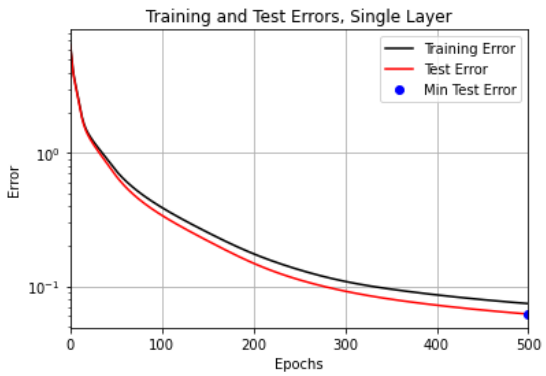


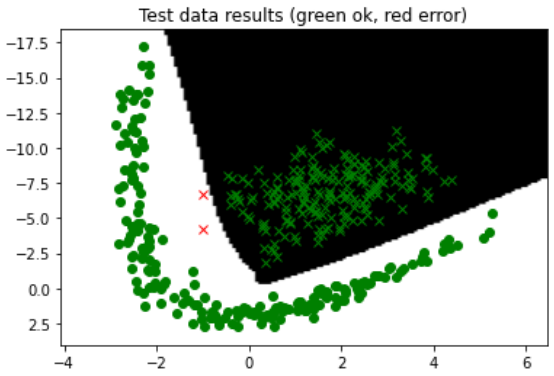
Dataset 2:

Hidden Neurons: 10
Number of iterations: 500
Learning rate: 0.02

A bit more complex feature space so therefore some more neurons than in the previous example. Tendency to stop learning and therefore we increased the learning rate to 0.02. For this we had to compensate by lowering the number of iterations.

Accuracy: 99.5%.



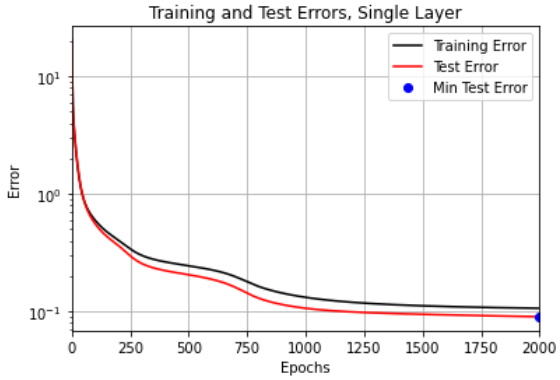


Dataset 3:

Hidden Neurons: 15.
Number of iterations: 2000.
Learning rate: 0.035

More complex → more hidden neurons. Stuck in local minima, therefore, we increased the learning rate. 2000 iterations seemed to be a sufficient number of iterations for the model to learn.

Accuracy: 99.75%.

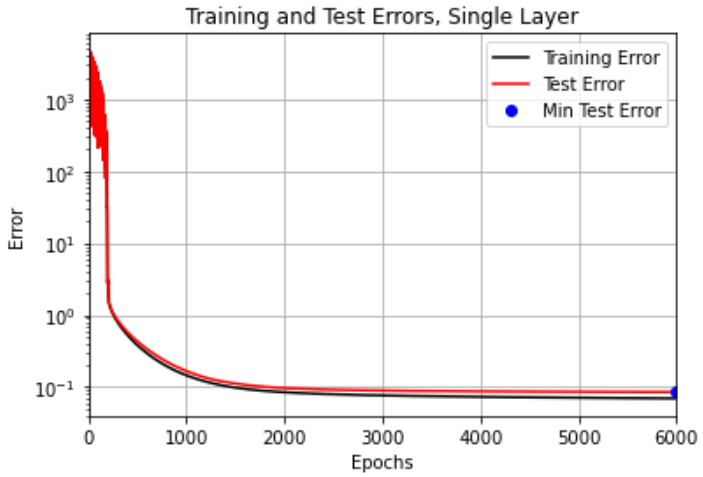


Dataset 4:

Hidden Neurons: 280
Number of iterations: 6000
Learning rate: 0.07

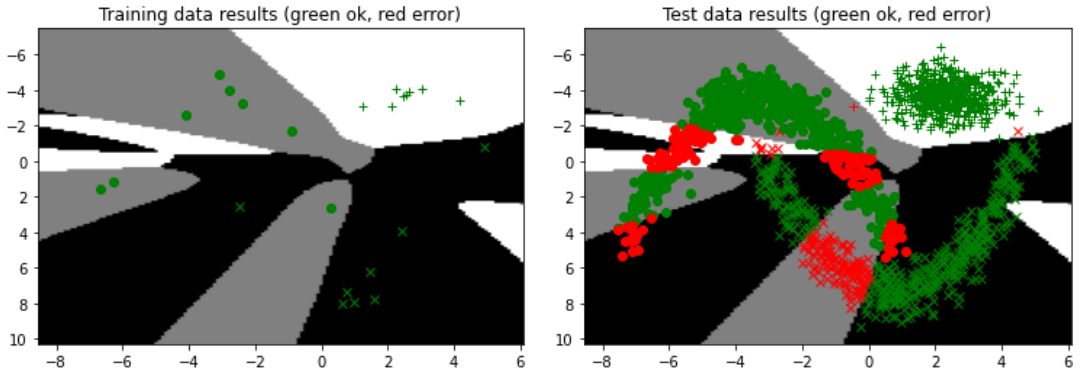
This one was a bit harder to tune, and as seen in the graph it's far from optimal but at least it reached 96% accuracy. As the dataset and task is far more complex than the other datasets we added a lot more neurons to the hidden layer, then we lowered the learning rate as we saw that the model ocellated very heavily, then we added enough iterations to reach 96% acc, so more iterations could maybe make it a little better but at the same it could start over fitting.

Accuracy: 96.00%.



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

Training: 100%, test: 84%



Because the small amount of training data can not represent the actual classes i.e the training data is not enough to see a clear pattern/cluster in the feature space**.**

A good observation one can do is look at the training data image and compare it to the test data image from Q7 (shown again here below) and see that the small amount of training data we chose for this task can't represent the "real" appearance of the class clusters.

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

Pros kNN: Simple model. Good results when classes are clearly distinguishable.
Cons kNN: Slow when large amounts of data.

Pros single layer network: It is a decent linear classifier. Fast computing
Cons single layer network: Often can't handle big and complex data because it can only solve linearly separable problems.

Pros multi-layer network: It is a decent non-linear classifier. Can handle high-dimensional input and many classes.
Cons multi-layer network: A bit more complicated. Overfitting can be a problem. Can be difficult to tune the hyper-parameters.

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

More layers would of course most likely improve the results, when it comes to algorithms and pre-processing some convolution for dataset 4 would probably improve the results. Likewise, normalization and standardization for all datasets could improve the results.