TBMI26 – Computer Assignment Report Supervised Learning

February 6 2018

Authors:
Jakob Dalsgaard, jakda997@student.liu.se
Erik Sandlund, erisa920@student.liu.se

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. If you meet the deadline we correct the report within one week after the deadline. Otherwise we give no guarantees when we have time.

1. **Give an overview of the data from a machine learning perspective. Consider if you need linear or non-linear classifiers etc.**

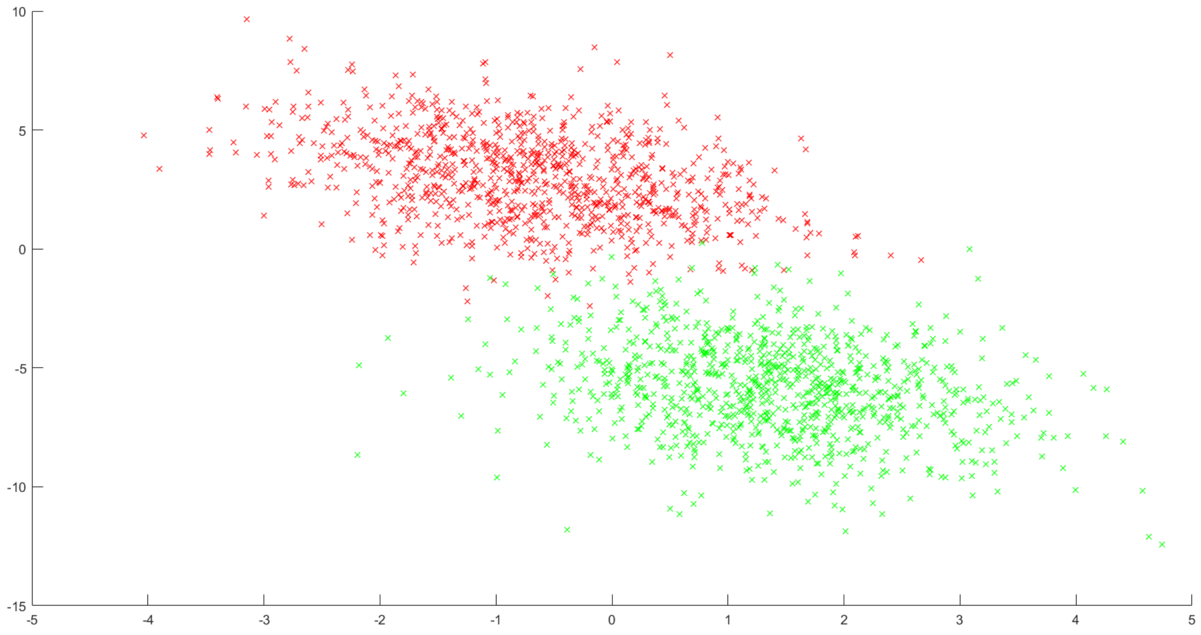


Figure 1: Data set 1.

Data set 1 consists of features of two classes, see figure 1. These are grouped together in quite a similar fashion and are, with the exception of a few outliers, easy to separate with a single line. So for data set, a linear classifier would suffice.

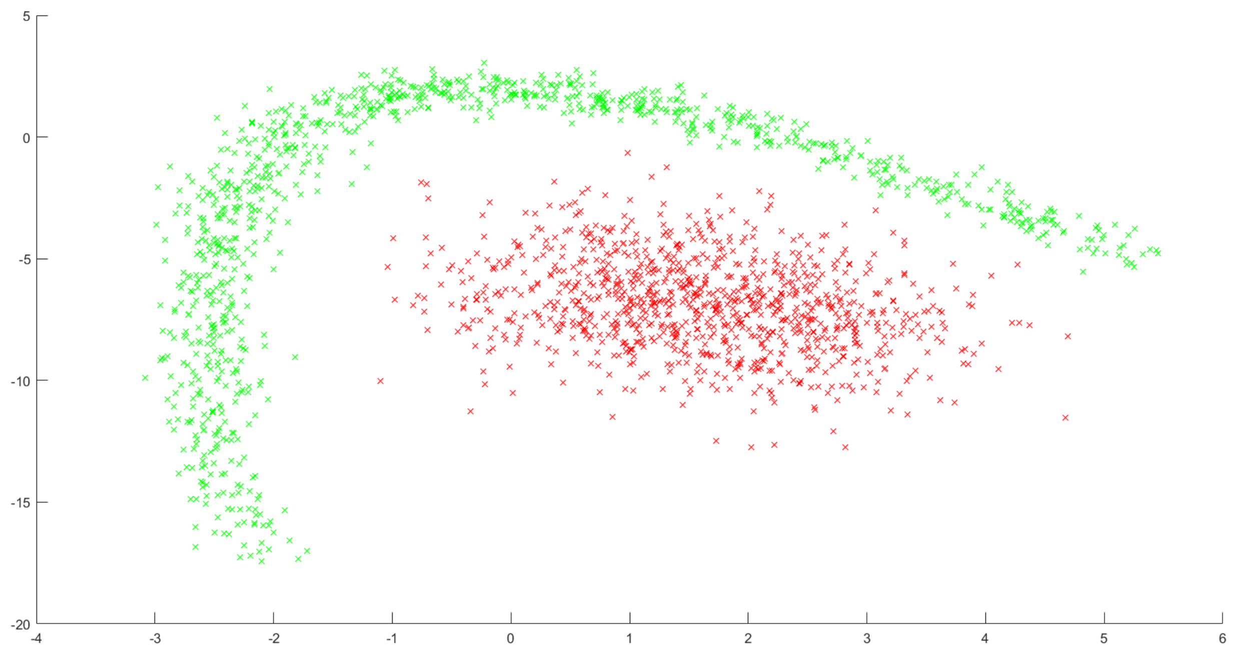


Figure 2: Data set 2.

The features in data set 2, see figure 2, do not have quite as many outliers in data set 1. These however are clearly not linearly separable, so a non-linear classifier would be preferable to classify this data.

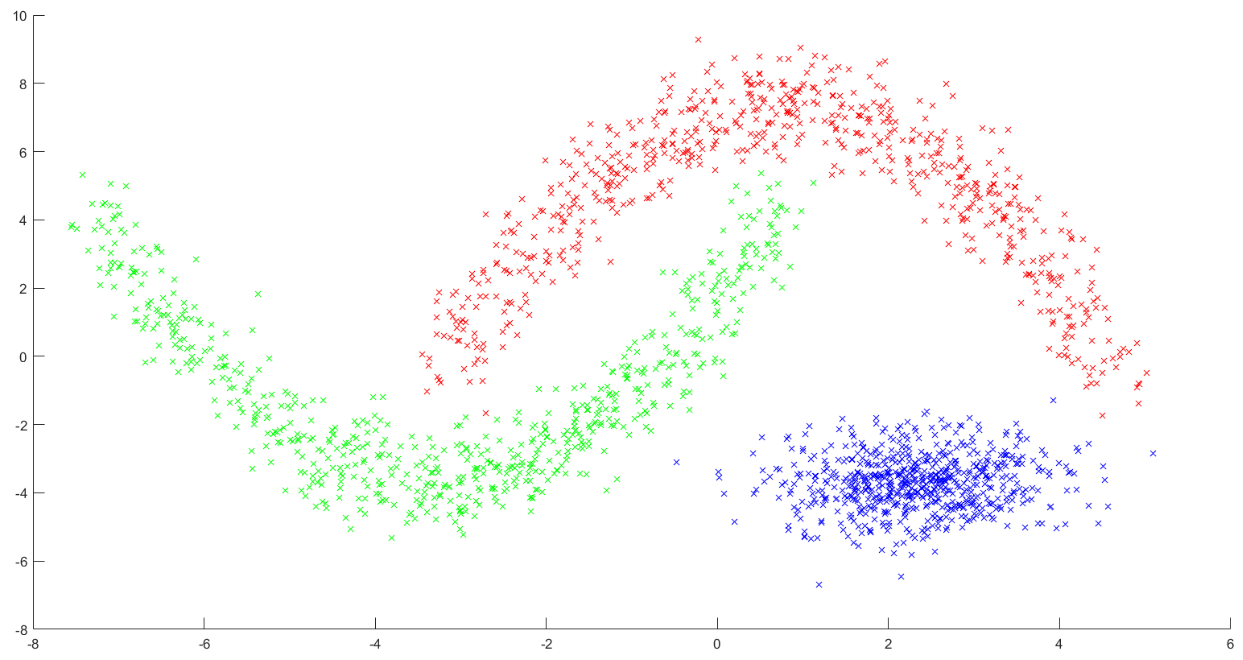


Figure 3: Data set 3.

This data set consists of features of three classes, see figure 3. These are non-linearly separable and thus a non-linear classifier would yield much better performance for this data.

The OCR data set has features of ten classes which are separable by no means other than a non-linear classifier.

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 OCR data is sampled down to drastically reduce the dimensionality of the complexities of the models necessary for classifying the data. It also gives invariance to small distortions in the data, such as noise.

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

We first use the Matlab function *pdist2* to calculate and set up a matrix containing the Euclidean distance from each point in the test data to all points in the training data. We call this matrix *distance*. Then we utilize the same function to create another matrix containing the k shortest Euclidean distances from each point in the test data to the points in the training data. These are automatically sorted in ascending order by *pdist2*.

We then take these two matrices enter a for loop that does the same thing for all points in the test data: We initialize the array *min_points*, to store the k shortest Euclidean distances for one point at a time and create an array *class_of_points*, which is to contain the classes of the k nearest points in the training data. Then we, for each of the k points that are closest to the test data point, find which index these points have in *distance* and compare this to *Lt* to see which class they have. We store the class of each nearest point in *class_of_points* and then use the *mode* function to see which class has the greatest number of nearest points.

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

The handling of draws in kNN is taken care of by the Matlab function *mode*, which in the case of a draw simply takes the one that has the least value. So if we have the case that one nearest point is of class 1 and the other is of class 2, the test data point will be classified as class 1, since 1 < 2.

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.**

The best k is found by dividing the data into three parts (this can be altered to get a n-fold cross validation for an arbitrary n). Then we initialize the values for the best k and the system accuracy for that k. After that we go through a for loop that tests the kNN algorithm with cross validation for a k from 1 to 9 (which can easily be altered if one wants to test up to another value for k). In the for loop, we initialize the constant *systemAcc*, which will eventually yield the mean accuracy for the system after the cross validation. Then we enter another for loop, which is the cross validation part of the code: for each of the parts the data set has been divided into, we initialize it as our test data and the rest of the data set as the training data and divide the label data set in the same fashion. We then run the kNN-algorithm and calculate the accuracy of the system for each part of the cross validation.

After we have run the cross validation, we calculate the mean accuracy and set *systemAcc* to it. If this is better than the previous best accuracy *bestAcc*, we set *bestAcc* to *systemAcc* and the previous best k to the current k. This is repeated this for all k that we want to iterate through. After all this, the best k and the accuracy it yields is printed.

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

For the single-layer, we set the training and test data to also contain the bias weights by simply concatenating a row vector of ones with the training data. We then set random weights in the interval [-0.01, 0.01] and start training the network by simply using the gradient optimization method. It is implemented as a linear classifier, with the output simply being the weight matrix times the input.

The multi-layer works in a similar fashion. The training and test data are also set to contain the bias weights in the same fashion as for the single-layer. The network is then trained, once again with the gradient optimization method. The big differences are that now there is a hidden layer as well, which will yield another gradient, and also that we now have hyperbolic tangents as activation functions for both the hidden and the output layers.

7. **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.**

The motivations for choosing each network can be found in the answer to question 1.

For the first data set, we used the single-layer network with 40 000 iterations and a step size of 0.0000005, which yielded an accuracy of 98.50 %, see figure 4 for the results.

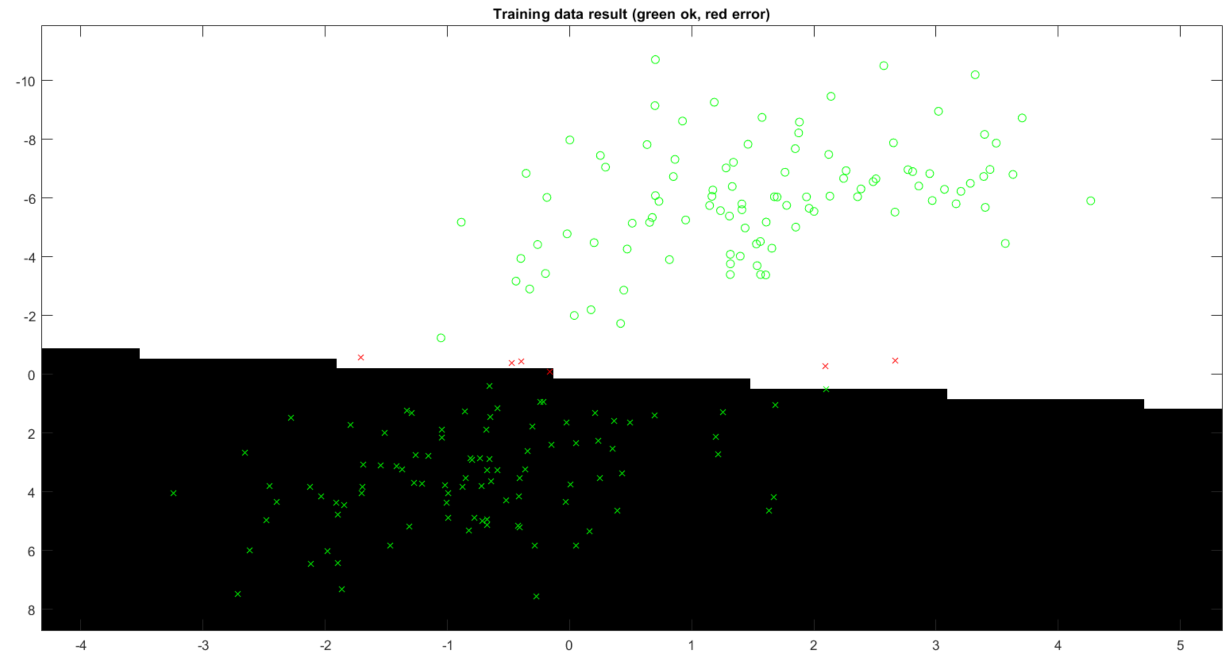


Figure 4: The test data result after running the single-layer network for dot cloud 1.

For the second data set, we used the multi-layer network with 8 hidden neurons, 4 000 number of iterations and a learning rate of 0.0025. This yielded a result with an accuracy of 99.10 %, see figure 5.

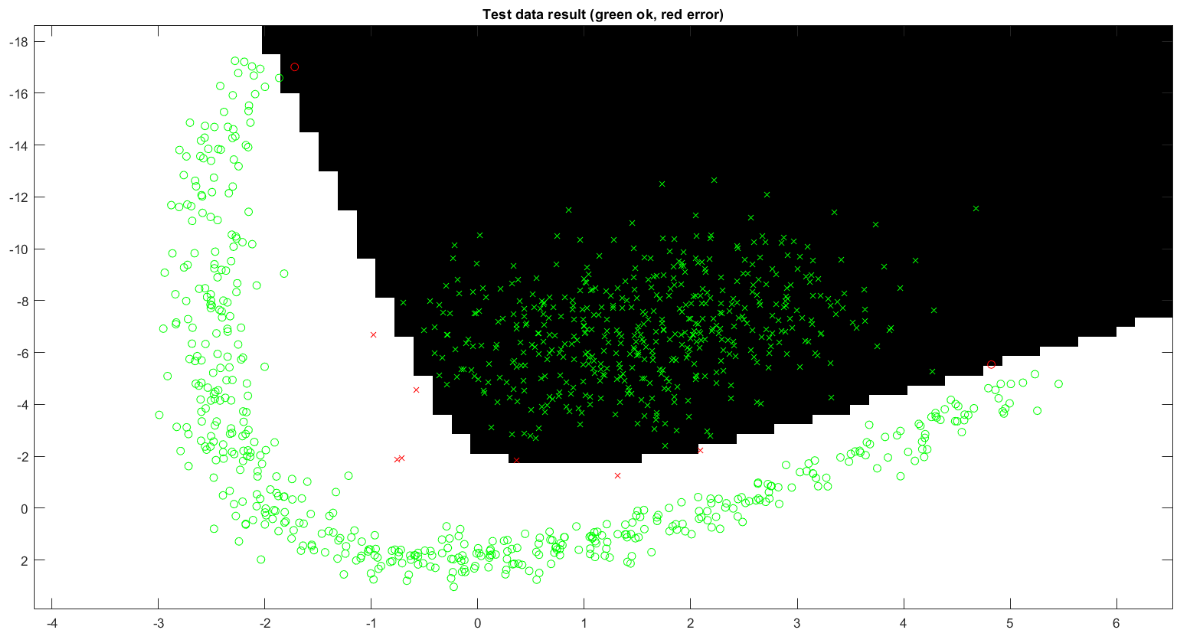


Figure 5: The test data result after running the multi-layer network for dot cloud 2.

For the third data set, we used the multi-layer network with 30 hidden neurons, 30 000 number of iterations and a learning rate of 0.0025. This yielded a result with an accuracy of 99.70 %, see figure 6.

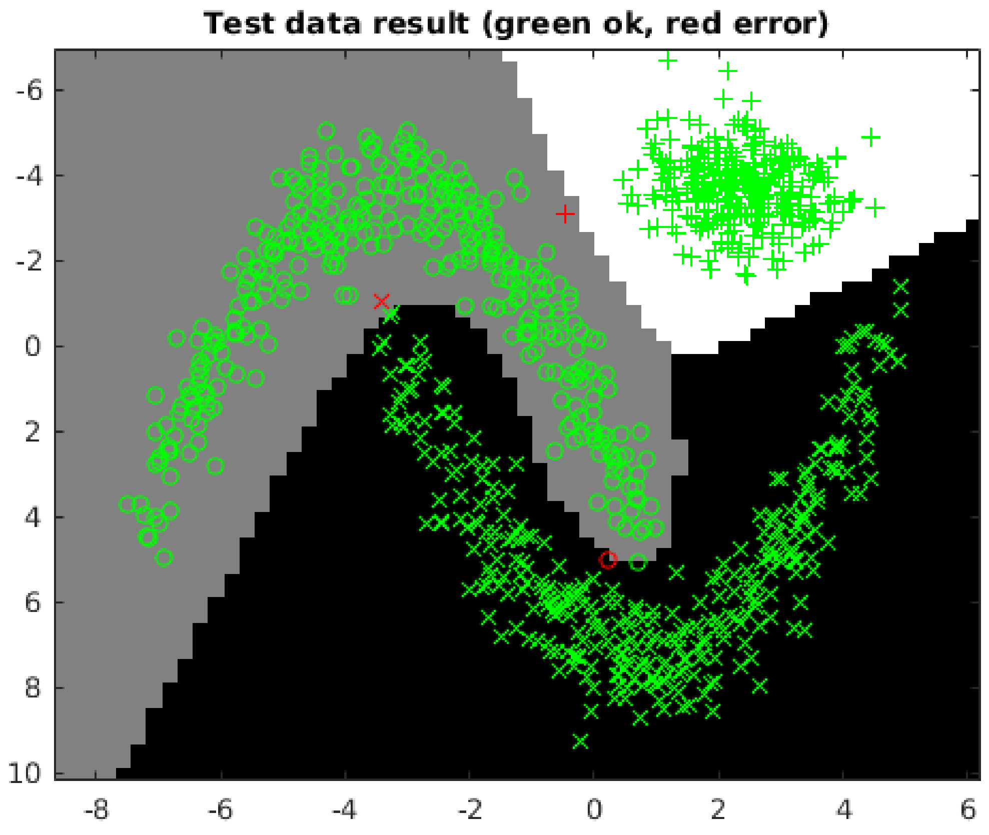
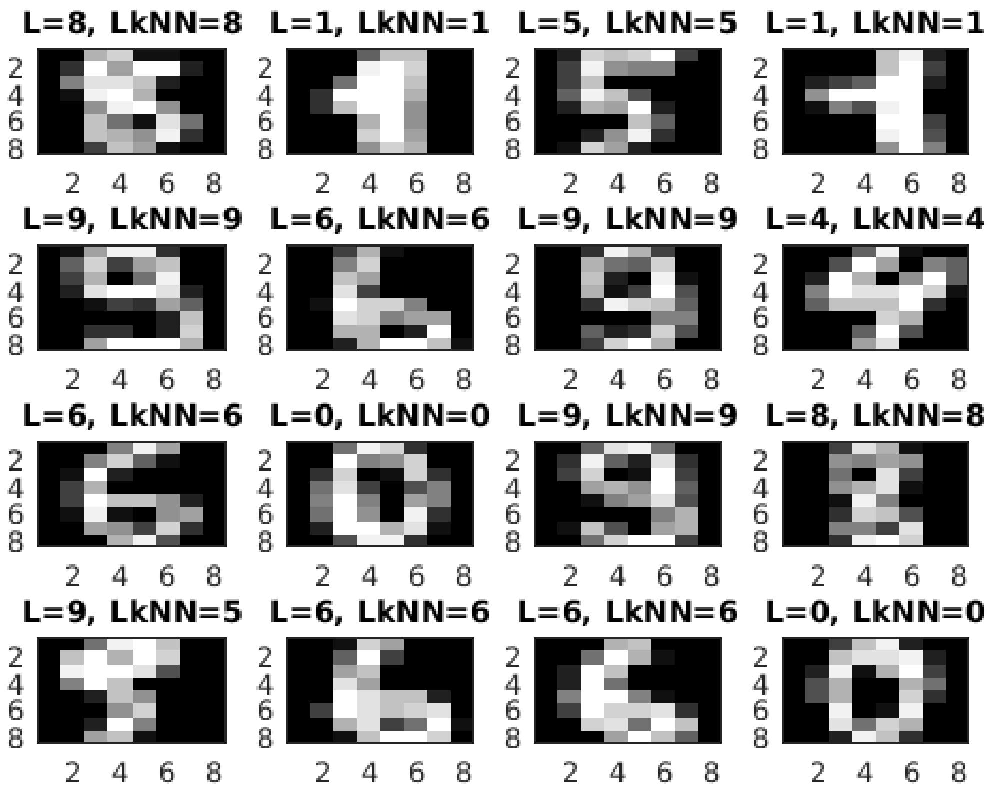
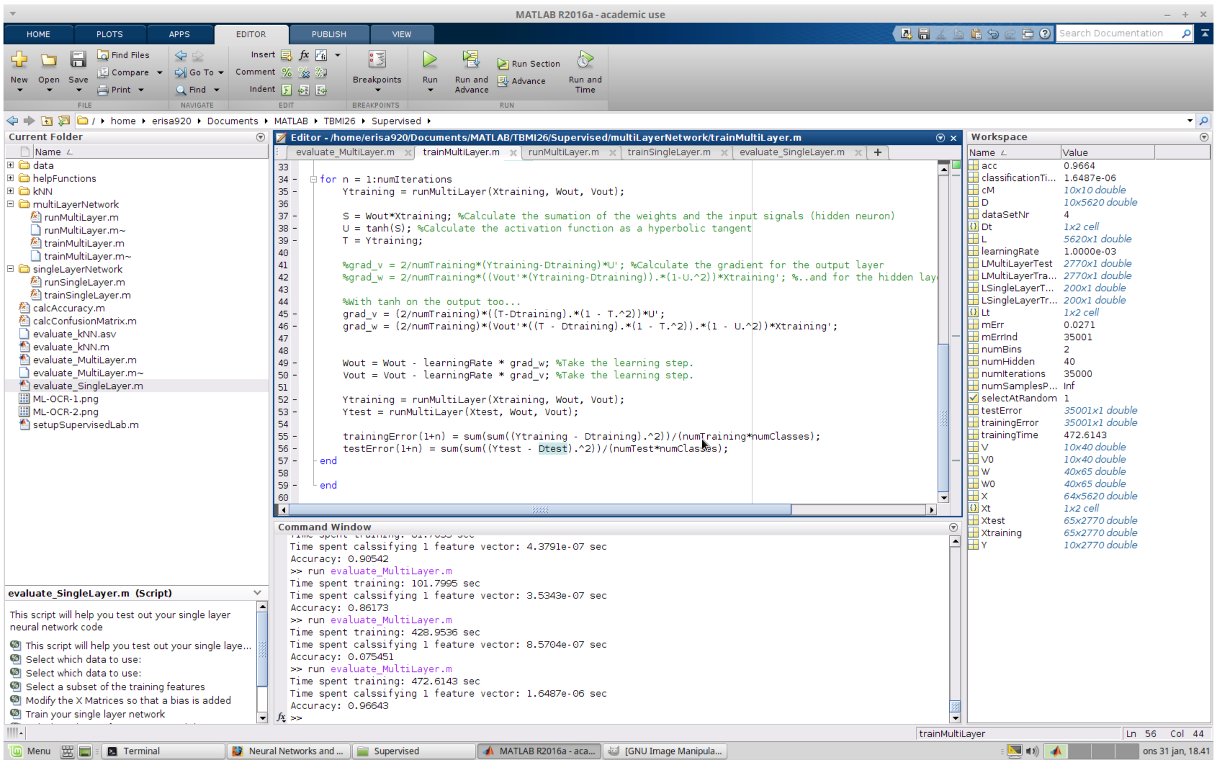


Figure 6: The test data result after running the multi-layer network for dot cloud 3.

For the OCR data set, which proved to be more difficult to classify with its 10 classes, we used the multi-layer network with 40 hidden neurons, 35 000 number of iterations and a learning rate of 0.001. This yielded a result with an accuracy of 96.643 %, see figure 7 and 8.





Current Folder

```
Name
⊞ data
⊞ helpFunctions
⊞ kNN
⊟ multiLayerNetwork
    runMultiLayer.m
    runMultiLayer.m~
    trainMultiLayer.m
    trainMultiLayer.m~
⊟ singleLayerNetwork
    runSingleLayer.m
    trainSingleLayer.m
  calcAccuracy.m
  calcConfusionMatrix.m
  evaluate_kNN.asv
  evaluate_kNN.m
  evaluate_MultiLayer.m
  evaluate_MultiLayer.m~
  evaluate_SingleLayer.m
  ML-OCR-1.png
  ML-OCR-2.png
  setupSupervisedLab.m
```

evaluate_SingleLayer.m (Script)

This script will help you test out your single layer neural network code

- This script will help you test out your single laye...
- Select which data to use:
- Select which data to use:
- Select a subset of the training features
- Modify the X Matrices so that a bias is added
- Train your single layer network

Editor - /home/erisa920/Documents/MATLAB/TBMI26/Supervised/multiLayerNetwork/trainMultiLayer.m

evaluate_MultiLayer.m | trainMultiLayer.m | runMultiLayer.m | trainSingleLayer.m | evaluate_SingleLayer.m

```matlab

for n = 1:numIterations
    Ytraining = runMultiLayer(Xtraining, Wout, Vout);

    S = Wout*Xtraining; %Calculate the sumation of the weights and the input signals (hidden neuron)
    U = tanh(S); %Calculate the activation function as a hyperbolic tangent
    T = Ytraining;

    %grad_v = 2/numTraining*(Ytraining-Dtraining)*U'; %Calculate the gradient for the output layer
    %grad_w = 2/numTraining*((Vout'*(Ytraining-Dtraining)).*(1-U.^2))*Xtraining'; %..and for the hidden lay

    %With tanh on the output too...
    grad_v = (2/numTraining)*((T-Dtraining).*(1 - T.^2))*U';
    grad_w = (2/numTraining)*(Vout'*((T - Dtraining).*(1 - T.^2)).*(1 - U.^2))*Xtraining';


    Wout = Wout - learningRate * grad_w; %Take the learning step.
    Vout = Vout - learningRate * grad_v; %Take the learning step.

    Ytraining = runMultiLayer(Xtraining, Wout, Vout);
    Ytest = runMultiLayer(Xtest, Wout, Vout);

    trainingError(1+n) = sum(sum((Ytraining - Dtraining).^2))/(numTraining*numClasses);
    testError(1+n) = sum(sum((Ytest - Dtest).^2))/(numTest*numClasses);
end

end

```

Command Window

```
Time spent calssifying 1 feature vector: 4.3791e-07 sec
Accuracy: 0.90542
>> run evaluate_MultiLayer.m
Time spent training: 101.7995 sec
Time spent calssifying 1 feature vector: 3.5343e-07 sec
Accuracy: 0.86173
>> run evaluate_MultiLayer.m
Time spent training: 428.9536 sec
Time spent calssifying 1 feature vector: 8.5704e-07 sec
Accuracy: 0.075451
>> run evaluate_MultiLayer.m
Time spent training: 472.6143 sec
Time spent calssifying 1 feature vector: 1.6487e-06 sec
Accuracy: 0.96643
>>
```

Workspace

Name	Value
acc	0.9664
classificationTi...	1.6487e-06
cM	10x10 double
D	10x5620 double
dataSetNr	4
Dt	1x2 cell
L	5620x1 double
learningRate	1.0000e-03
LMultiLayerTest	2770x1 double
LMultiLayerTra...	2770x1 double
LSingleLayerT...	200x1 double
LSingleLayerTr...	200x1 double
Lt	1x2 cell
mErr	0.0271
mErrInd	35001
numBins	2
numHidden	40
numIterations	35000
numSamplesP...	Inf
selectAtRandom	1
testError	35001x1 double
trainingError	35001x1 double
trainingTime	472.6143
V	10x40 double
V0	10x40 double
W	40x65 double
W0	40x65 double
X	64x5620 double
Xt	1x2 cell
Xtest	65x2770 double
Xtraining	65x2770 double
Y	10x2770 double

Figure 8: The matlab UI after running testing the OCR data

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

We chose to set the *numSamplesPerBin* to 15. Using a multi-layer network with 40 hidden neurons, 35 000 number of iterations and a learning rate of 0.001 yielded the results seen in figure 9, 10 and 11.

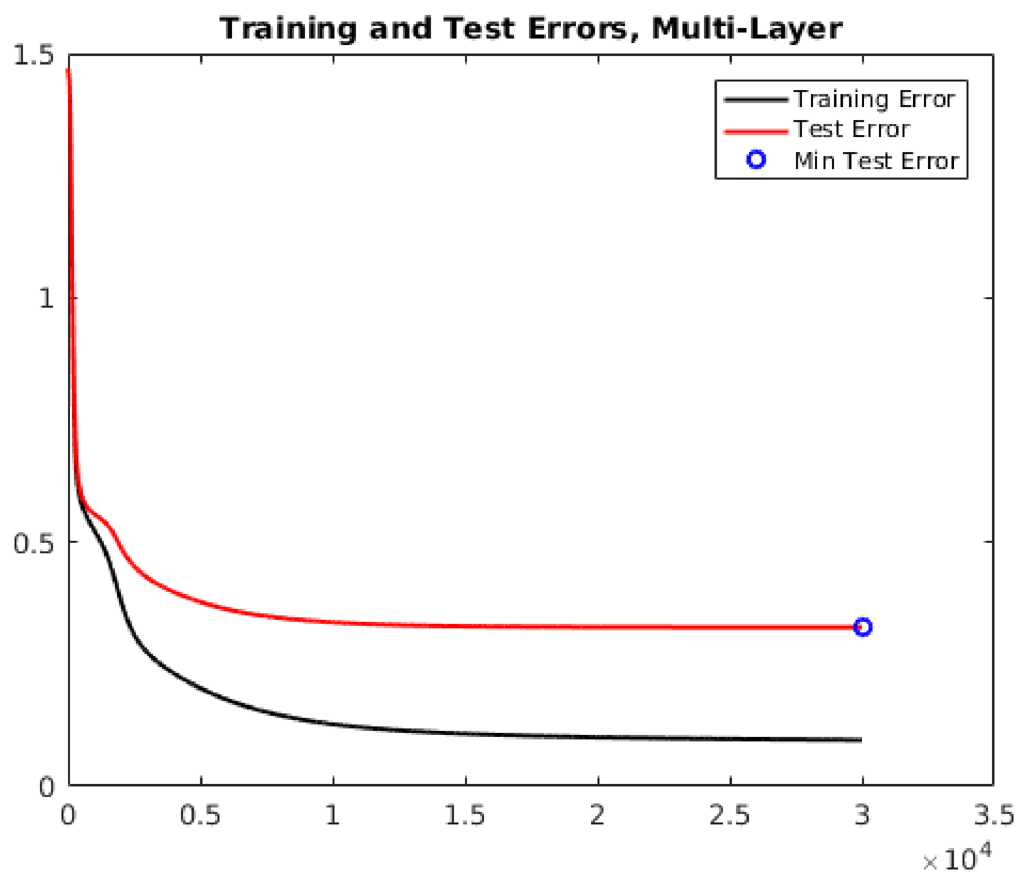


Figure 9: Training and test error for our non-generalizable solution.

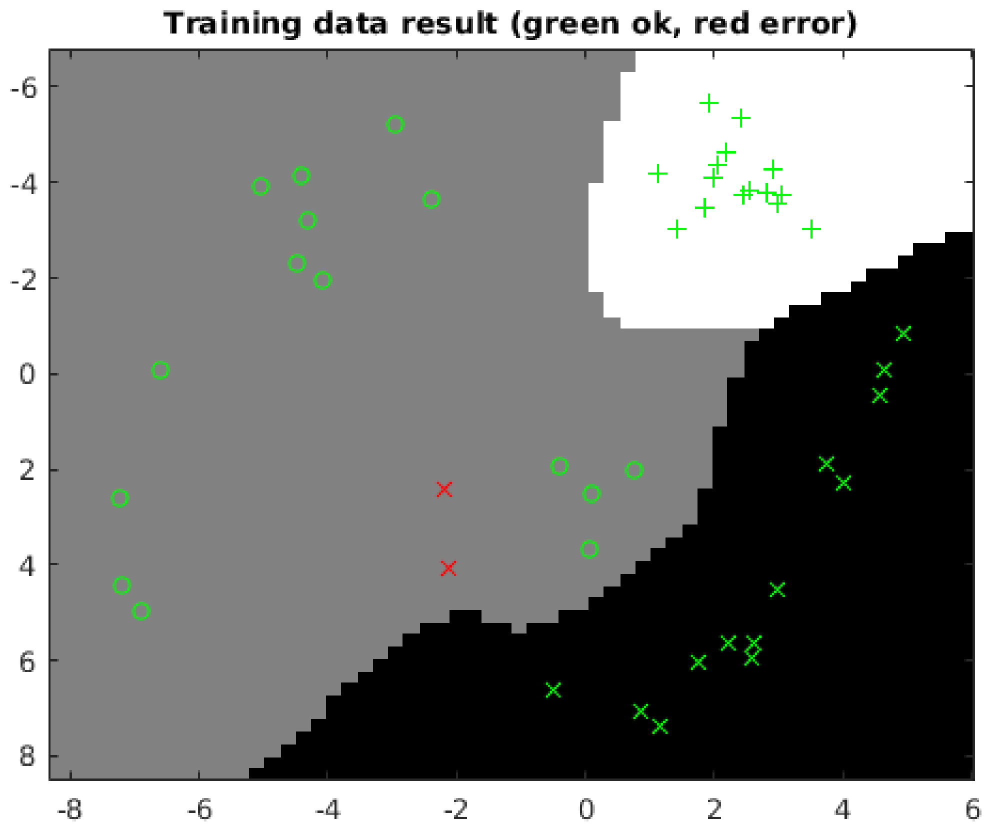


Figure 10: The training data result for our non-generalizable solution.

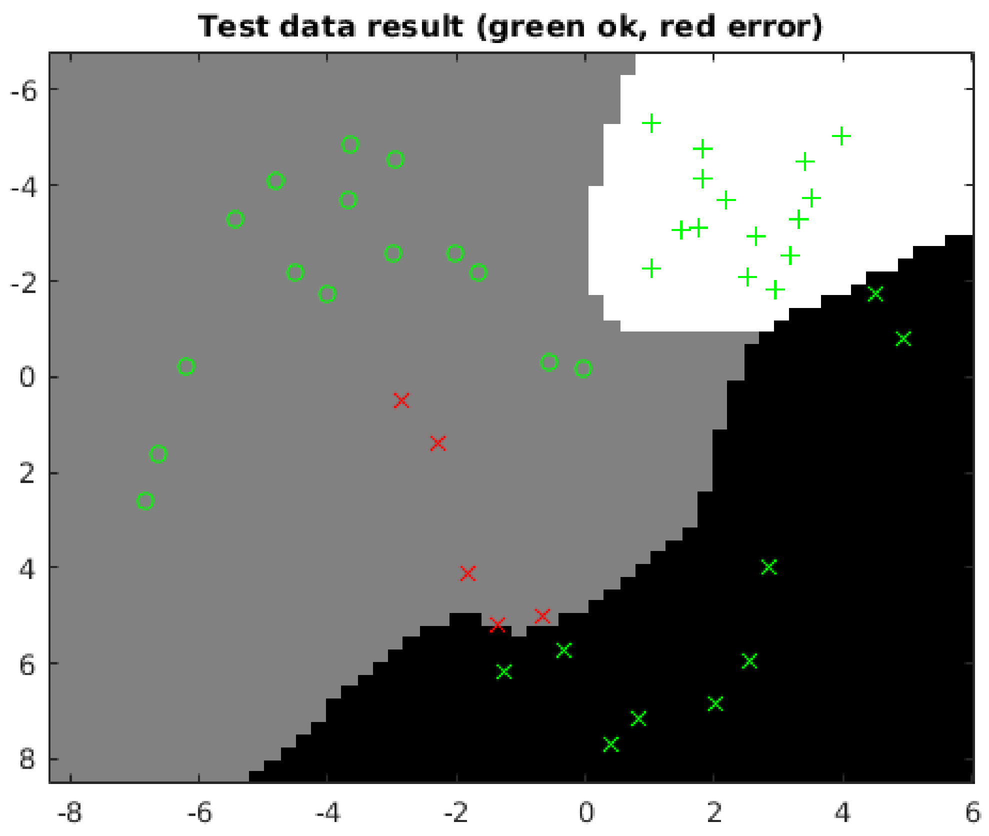


Figure 11: The test data result for our non-generalizable solution.

When you look at the figures, it is clear that this is a classic example of overfitting. There are too little data and the model is far too complex to classify the data correctly with these number of sample features. In short, with too little data, a fair representation of the classes is not given in the training data and thus it will be unable to classify it. See for instance in figure 10 where the circles to the lower right will make sure that the crosses to the upper left in figure 11 will never be classified correctly.

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

The kNN algorithm has an advantage that it can classify features of classes that are non-separable. It also does not require any training of a model. However, it is easily affected by outlier clusters and can be computationally expensive. It also requires us to save every observed sample for reuse if we want to classify a new sample.

Single-layer networks are in general much less computation-heavy and simple, so they are preferable for classifying features of linearly separable classes. They are however quite useless at classifying anything other than features of linearly separable classes, which makes them quite niched in its usefulness and generally, it is not an adequate type of classifier.

Multi-layer networks, on the other hand, have the advantage that they can classify much more complicated classes, but for complex classes an equally complex model is necessary and thus they can become very computation-heavy. Its parameters must also be finetuned to get a good result (i.e. find the necessary number of hidden neurons etc.), which can be quite time consuming.

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

The kNN algorithm could for instance be improved by determining the tie break by calculating which class has the least average distance to the point in question.

The parameters in the multi-layer network could probably be finetuned even further to have more optimal performance, for instance by doing cross-validation. This could however take a lot of time.