Erik Linder-Norén
Olle Nilsson TBMI26 2017-02-07

Supervised learning: kNN and Backpropogation

Data analysis:

To determine the type of classifier needed to best classify the different classes of the datasets. For **dataset 1** we see that there are two gaussian distributed classes. Data from both distributions are overlapping between the classes and it will therefore pretty much impossible to find a perfect separation that does not overfit. For this dataset a linear classifier would suffice.
In **dataset 2** we see that there are two classes which are perfectly separable using a nonlinear decision boundary. For this problem we would get a much better result using a nonlinear classifier than a linear one. In **dataset 3** there are three classes perfectly separable using a nonlinear decision boundary. A nonlinear classifier would give as a much better result than a linear one.
In **dataset 4** we have 10 classes which are only separable with nonlinear models.

OCR downsampling:

The purpose of downsampling the OCR-data is to greatly reduce the dimensionality of the data and the complexity of the models needed to classify the data. The downsampling also gives invariance to small distortions in the data such as noise.

kNN:

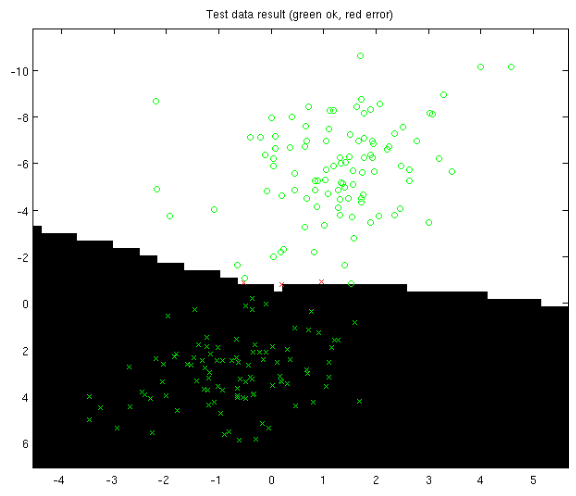
We begin by iterating through all of the points in the data and measure the distance from the class that we want to classify. We save the euclidean distance to each points and the corresponding label. When we have gone through the whole list of data points we sort the list containing the distances and labels by the distance value. We then pick the k first elements and uses the corresponding labels to those points to determine the class of the sample that we want to classify.

Tie breakers:

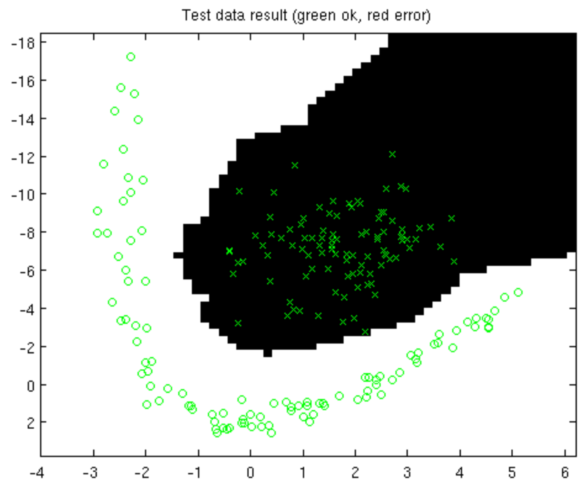
If we have a tie between two classes when we want a vote among the closest neighbors we classify the point as the class of the closest point to the one that we want to classify. A way that would probably be better is to choose the class of the two groups that has the lowest average distance to the sample. However, we decided to keep our solution which also seemed to work fine.

Cross validation:

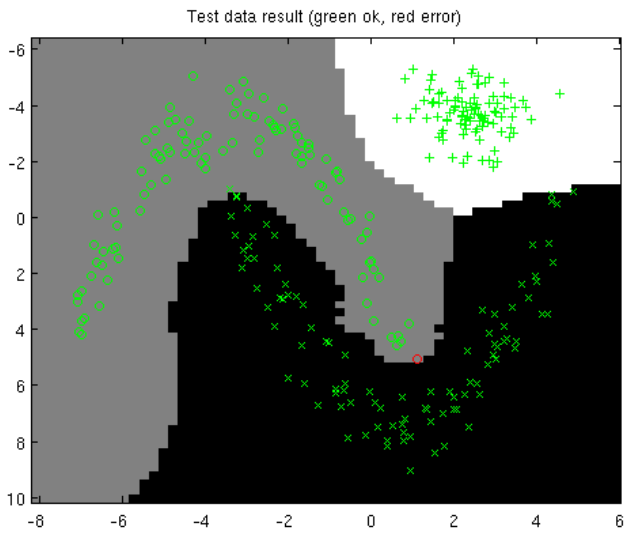
We decided to use three bins when measuring the cross validation scores for different k values. We iterated through k values from 1 to 7 and compared the cross validation scores for the different k values and chose the one with the highest accuracy.



Dataset 1



Dataset 2



Dataset 3

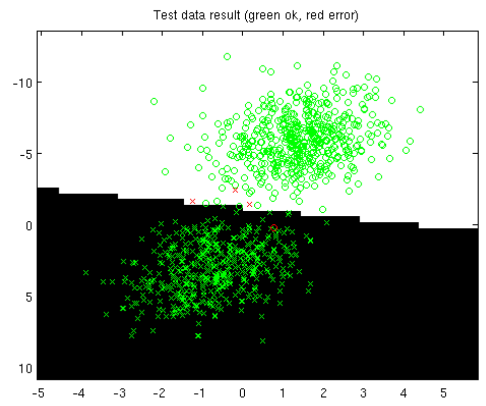
Backpropogation Networks:

Perceptron:

We added a one as the first feature of each training sample to take the bias weights into account. We initialized the weights in the range (-1/sqrt(N), 1/sqrt(N)), where N is the number of features of the samples. We then updated the weights in the output layer for a specified number of times and the made the prediction with the resulting weights. As the activation function we used the hyperbolic tangent.

Multilayer perceptron:

We added a one as the first feature of each training sample to take the bias weights into account. We initialized the weights of the hidden layer and the output layer in the range (-1/sqrt(N), 1/sqrt(N)), where N is the number of features of the samples. We then updated the weights in the output layer for a specified number of times and the made the prediction with the resulting weights. As the activation function of the hidden layer we used the hyperbolic tangent. We also added one as the first row value of the output of each neuron in the hidden layer so we could use bias weights in the output layer as well.



Dataset 1:

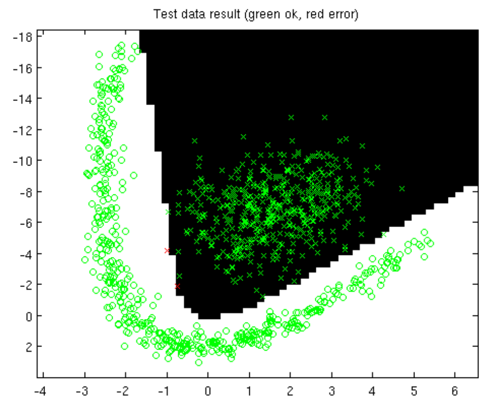
Accuracy: 0.996

Hidden neurons: 1

Iterations: 2000

Learning rate: 0.01

Since the data could be separated using a linear classifier just as well as with a nonlinear one we decided to keep the number of neurons low and also the number of iterations. Adding neurons and iterations did not improve the accuracy by much.



Dataset 2:

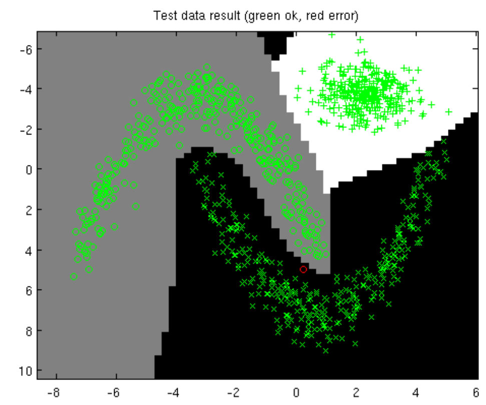
Accuracy: 0.998

Hidden neurons: 8

Iterations: 2000

Learning rate: 0.01

As we would need a nonlinear classifier to separate the classes in this dataset we tried more to use more neurons compared to the first dataset and found that 8 hidden neurons gave us good results.



Dataset 3:

Accuracy: 0.999

Hidden neurons: 8

Iterations: 8000

Learning rate: 0.01

For dataset 3 we kept the number of neurons at 8 and let the model train for 8000 iterations and this gave us a nice boundary between all classes.

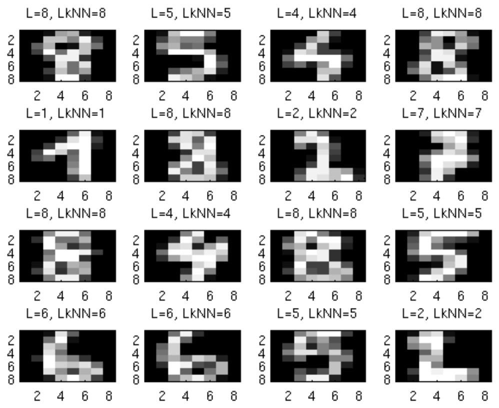
Dataset 4:

Accuracy: 0.971

Hidden neurons: 40

Iterations: 8000

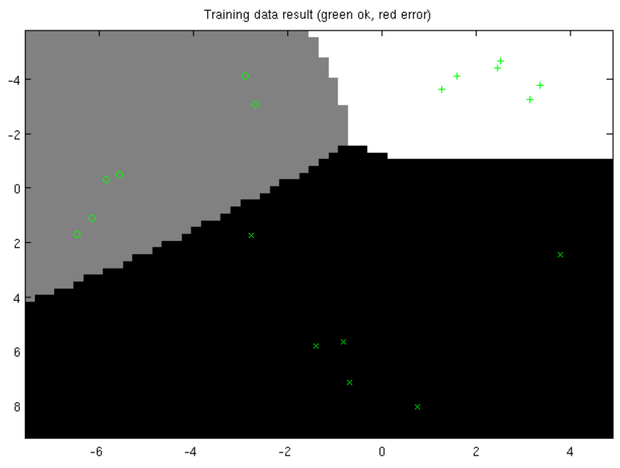
Learning rate: 0.01

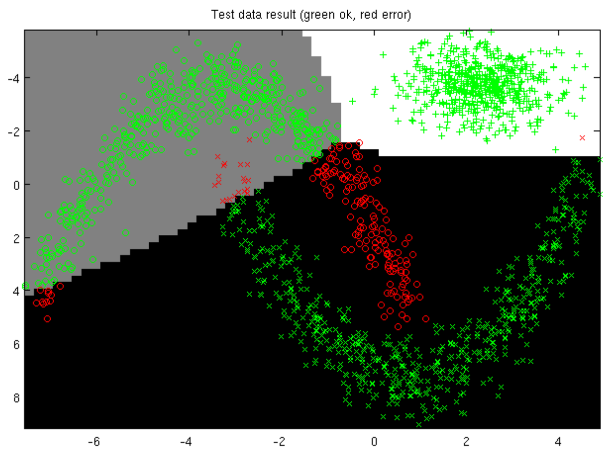


It was evident early that we would need a more complex model to correctly classify the digits in this dataset as we got a accuracy score of around 0.5 with 8 hidden neurons. We added neurons until we got a score above 0.96. This felt reasonable since we needed to separate 10 classes from each other. Because of the fact that we needed a more complex model we needed 8000, or more , training iterations for the model to fit the data well.

Non generalizable:

To accomplish this we increased the number of bins to 100 and trained the model only on the first bin while finally being tested on all of the other bins combined. We increased the number of hidden neurons to 200 and the number of iterations to 10000 to fit the model to the small amount of training data that it had to train on. This of course did not give good result on the larger amount of test data that it later had to predict classes for, as it had overfit the training data.
We could also let the model train on a very small set of training data with a low complexity to underfit the training data and the use that model to make a prediction on the test data. This would also generalize very badly.





Discussion:

The kNN classifier generally gives a good approximation since it can do nonlinear classification by proximity to observed samples, and does not require that we train a model. However, it is somewhat computationally expensive and require us to save all of the observed samples for reuse every time we want to classify a new sample.

The perceptron is a very simple model that can do linear classification but is not able to do nonlinear classification. This gave us a good result on dataset 1 for example as a linear decision boundary gave us as good of a result on that dataset as a nonlinear one.

The multilayer perceptron required us to tune its hyper parameters and a good enough number of training iterations for it to converge to a good decision boundary and classification of the data. The problem with this model is that it is generally hard to know exactly how many neurons, how many iterations and the learning rate that will give the best results. A multilayer perceptron with at least one hidden layer has been proven to be able to approximate any function and that is great strength of this model.

Possible improvements:

One improve that can generally been done to improve the results of the multilayer perceptron is to choose the hyper parameters by doing cross validation. For the kNN we could handle tie breakers in a more robust way by taking the two conflicting classes average distance to the sample point and determine the samples class by the closest distribution.