Assignment 1 for Neural Network and Learning Systems

Hao Chi Kiang (haoki222), Yumeng Li (yumli241)

February 6, 2017

Contents

1 Classification using K-Nearest Neighbour

1.1 Overview of the data and linear separability

The first data set looks like a mixed Gaussian distribution with two seperate underlying Gaussain varaibles. It is very linearly seperable according to our sample. The second data set does not look like linearly seperable, however, it seems that two linear functions which form a V-shape can separate the two classes reasonably well. However if a model has to be chosen, I would not recommend linear model at all. The third data set consists of three classes, thus in the implementation of any methods following, we cannot assume binary classes. The third data set is obviously not linearly separable. However in the third data set, the data points in our sample forms three very clearly-cut clusters, meaning that high accuracy can be done at least using

only the observed data set. The relatively large sample size of the third data set also suggests that generalization of the model is not much of a problem, given the data set at our hand is a random sample. The fourth data set consists of 65 features, each of which is from a space of 17 integers, from zero to 16. They contain compressed information of pixels from images of handwritten digits. There are ten classes, each of which corresponds to an Arabic digit. Although we have no evidence regarding whether it is linearly separably or not, nor did we managed to plot a 65-dimensional Euclidean space, we see no reasons to assume such a complex data set to be simply linearly separable: not at least without some transformation of the features.

1.2 Down sampling of the OCR data set

In audio signal processing, down sampling refers to discarding some sampled points so that noises in small scale does not affect too much in the following signal processing. In our case of image processing, a high dimensional picture usually contains a lot of data which is irrelevant to our classification. For example, in a fine-grained, pixel-wise level, a dull pencil will produce a very different texture than a sharp pencil or a oil marker, even if the shape of the digit is similar. To address this problem, the data set was cleaned, according to the source web page, in a way that the original picture of dimension 32x32 is cut to 64 4x4 pieces. An average within each 4x4 pieces is taken (each pixel is either completely black or completely white, that is, either zero or one), producing in total 64 integers. Each one of these integers signifies the 'blackness' in its entire 4x4 piece. In other words, by the time we feed the data to our algorithm, the noise within each of the pieces has been dropped.

1.3 Summary of implementation of the kNN algorithm

We have chosen to use the simplest method that requires the calculation of all the distances between all the data points, since the size of our data set is not too large, and this $O(n^2)$ calculation of the distance matrix is pragmatically acceptable. (In cases where $O(n^2)$ distance matrix calculation is not feasible one may need to use algorithms such as the VP tree instead.)

Hence the first step in the algorithm is simply a nested for-loop which produces the distance matrix. Euclidean distance is chosen, since we don't see any reason to use one of the other metric functions. This results in a matrix which we could sort by columns to obtain the indexes of the K training data points which lies the nearest to each of the testing data points. For each testing data point, the proportion of each classes in its K nearest

training data point is calculated as taken as an estimate of the probability that the testing data point lies in the corresponding class.

1.4 Scenario in which K=2

Given a testing data point, if both of its two neighbours are in class A (resp. B), then we classify the testing point as class A (resp. B). When one of the neighbour is from class A and another class B, however, we do not intentionally prefer class A or B in our particular implementation. As soon as we ensure that our program terminates correctly with one of the class as output, we simply let the program to choose one. One may argue that it is better explicitly write a random draw from the two training points to guarantee the 'unbiasness' or 'fairness' of the classification, however, in practice we see no urgent need to do so: simply choosing an odd K would solve the problem. It is also worth noting that some rigorous machine learning textbook, on the other hand, assumes that there is a prespecified 'preferred' class when this happen. This again shows that there is a lot of wiggle room regarding this problem and it is an open question depending on application.

1.5 Cross validation

Figure 1 shows the mean accuracy using cross validation for each data set with different K's ($K \geq 2$). We can see that as K increases, the accuracy of classification does not changes so much. Slight decrease in accuracy is seen when K gets unnecessarily large. This is expected since our data are very good separated from the beginning. We would choose $K = 3$ for each data set, according to the graph.

1.6 Summary of back-propagation neural network

In the single layer version, the implementation is exactly the same as a simple linear regression, especially in our case the activation function for this layer is an identity function. A back-propagation is then done to update the coefficients (weights) according to the resulting error. This is done repeatedly in a loop, so that weights are updated many times. This is in fact the same as the last layer of the multi-layer version.

In the multi-layer version, we have an additional hidden layer. When a training data data point is fed into the network, a hidden activation vector is calculated from a linear combination of training features applied to an activation function, which in our case is the hyperbolic tangent function. If the number of hidden neurons, which is pre-specified beforehand, is K, there

will be K values in the hidden activation vector. This vector is then fed into the last layer.

The above computation will result in a prediction for the training data point. We will then calculate the prediction error (residual) for this prediction, and then use it to update the weights in the previous two layers using the method of gradient descents. The formula for the gradient descents update for the two layers are different.

1.7 Results from the neural network

We have implemented two slightly different versions of multi-layer neural network: one with a fixed learning rate, another with learning rate cut one-eighth whenever it detects the testing accuracy stopped decreasing. The former was used to classify data set number one to three, and the latter was used in data set four. The reason for using an adaptive learning rate in the fourth data set is that it gives much more stable accuracy, and much easier to find the right parameters.

We have reached the accuracy criterion for the first data set using $numHidden = 2$, $numIteration = 100$, and $learningRate = 0.0005$, with initial weights initialized as uniformly distributed numbers between zero and one. Since this data set is linearly separable, high accuracy is very easy to reach. Figure 2 and 3 shows the result of the classification for this data set. The accuracy for this run was 0.993.

For the second data set, the exactly same parameters are used and it gives very good and stable accuracy as well. Although this data set is not linearly separable like the first one, it consists of two very separated clusters and the boundary between the two clusters are very simple in shape: this is a sign that small number of neurons can learn it well. Figure 4 and 5 shows the result of the classification for this data set. The accuracy for this run was 0.996.

The third data set has three classes and more complicated boundaries between each classes, however, they are nonetheless very well separated. Thus we would expect good accuracy. The parameter we have used, that gives stable high accuracy, are $numHidden = 6$, $numIteration = 1000$, and $learningRate = 0.0001$, with initial weights initialized as uniformly distributed numbers between zero and one. Note that the learning rate used in this data set is five times smaller than previous one, and this helps the neural network to learn a more complicated boundary. A high learning rate generally results in a more 'rigid' boundary. Figure 6 and 7 shows the result of the classification for this data set. The accuracy for this run was 0.998.

When trying to learn the fourth data set with a fixed rate, we could find parameters which occasionally give accuracy higher than 96%, but they were not stable. We found that smaller learning rate gives better results, however, when the learning rate is too small, it has difficulty converging at the beginning of the training process. We then found that it helps to run the network with a big learning rate for a few loop first, then learn it again with a smaller rate and weights initialized to be the result of the previous run. However, this makes the choice of parameters more difficult: we now have to choose two learning rates, instead of one, so the space of parameter that we need to search has exploded twice as large. The solution to this problem is, as mentioned, using an adaptive learning rate scheme: at each $300n$ -th iteration, where n is a natural number, it calculates the amount of test error deceased since last time it checked. If it did not decrease more than some percentages (around 0.001% was used), it cuts the learning rate one-eighth.

The actual combination of parameters used in this data set was $numHidden = 256$, $numIteration = 2900$, and $learningRate = 0.000048$, with initial weights initialized as uniformly distributed numbers within $[-0.1, 0.1]$. Initial weights was carefully chosen so that the range is not too wide, which results in difficult/slow convergence, nor too narrow, which usually results in lower accuracy. Figure 8 and 9 shows the result of the classification for this data set. The accuracy for this run was 0.96462.

1.8 Non-generalisable solution for the third data set

Figure 10 and 11 are a visualisation of an overfit scenario for the third data set. In this run, only 25 data points were used in training, and parameters are chosen at $numHidden = 128$, $numIteration = 14000$, and $learningRate = 0.00013$, with initial weights initialized as uniformly distributed numbers within $[-0.1, 0.1]$. The high amount of hidden neurons make the model very flexible, and obviously 25 training data points are not enough for training a generalisable model.

1.9 Comparison between kNN and neuron network

Talking about our four data set only, we found that kNN is much easier to use, especially when tuning the parameters. However, kNN is not as scalable as neural network, since the running time of classification of neural network is invariant as the number of training points increases. Another advantage of neural network is that it is more flexible in a way that many aspects of the algorithm can be tuned to fit different applications: our tuning of using

an adaptive learning rate is an example.

1.10 Possible improvement

For kNN, the performance can obviously be improved: the implementation we used requires calculation of all the distances between all the data points, and this does not need to be the case. In fact, we have implemented a faster version of kNN, using the technique of 'VP-tree' (in the file Supervised/kNN/kNN2.m). The VP-tree implementation is a magnitude faster than the original one, especially when applied to the fourth data set.

For the neural network, we think that a more crafted learning rate scheme is indeed possible: the adaptive learning rate scheme that we have used in the fourth data set was quite improvised by trial-and-error. Also, the stopping criterion for the neural network can be made adaptive as well: this would save us some manual trials.

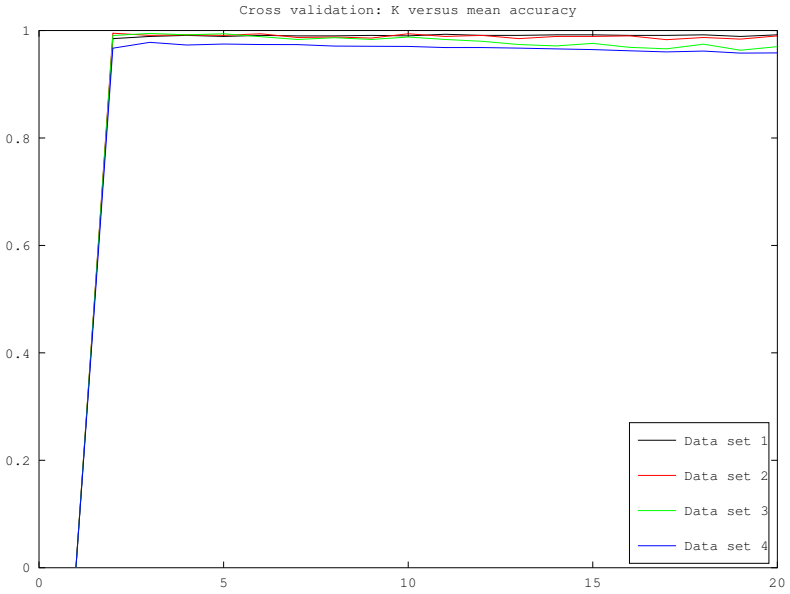

Figure 1: Cross validation result for kNN

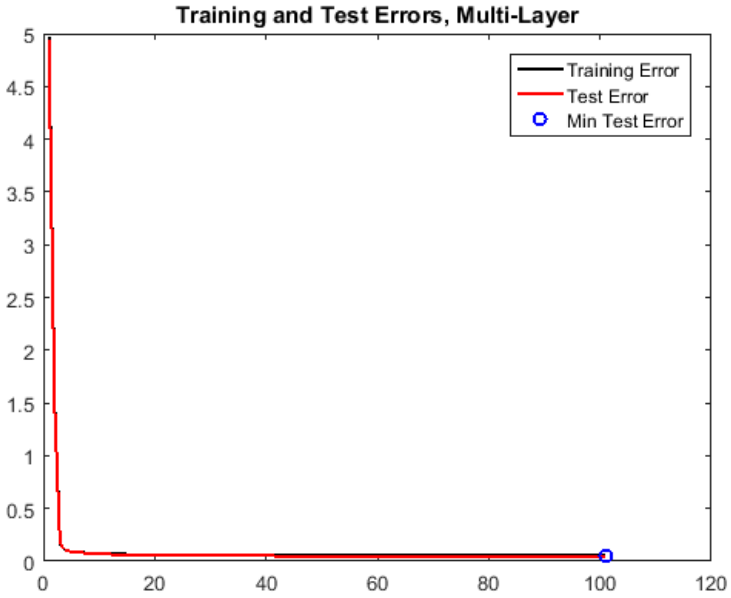

Figure 2: Error of the first data set

Figure 3: Learned boundary of the first data set

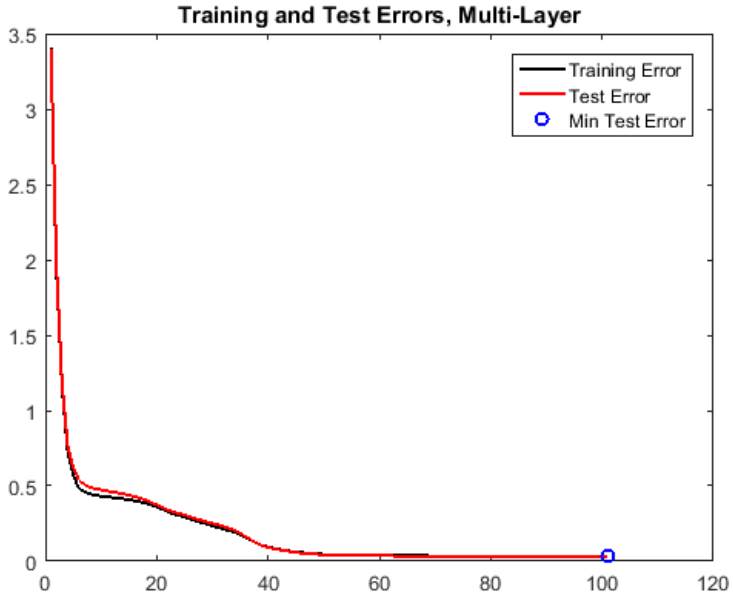

Figure 4: Error of the second data set

Figure 5: Learned boundary of the second data set

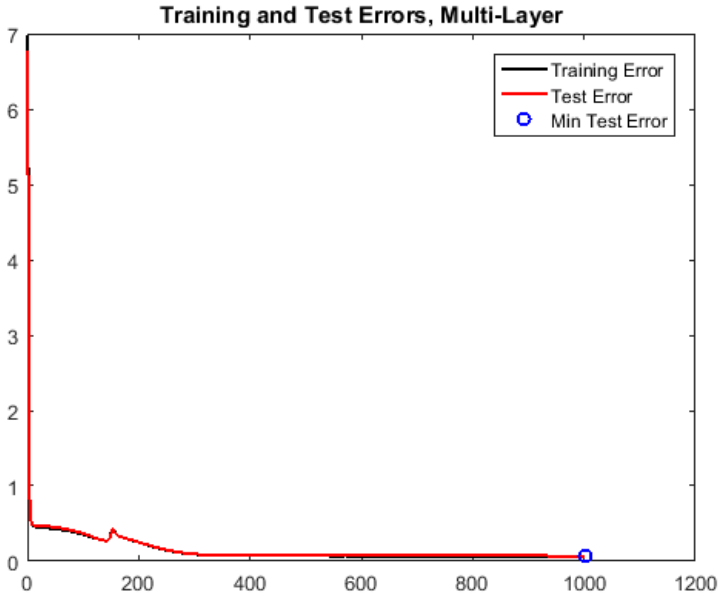

Figure 6: Error of the third data set

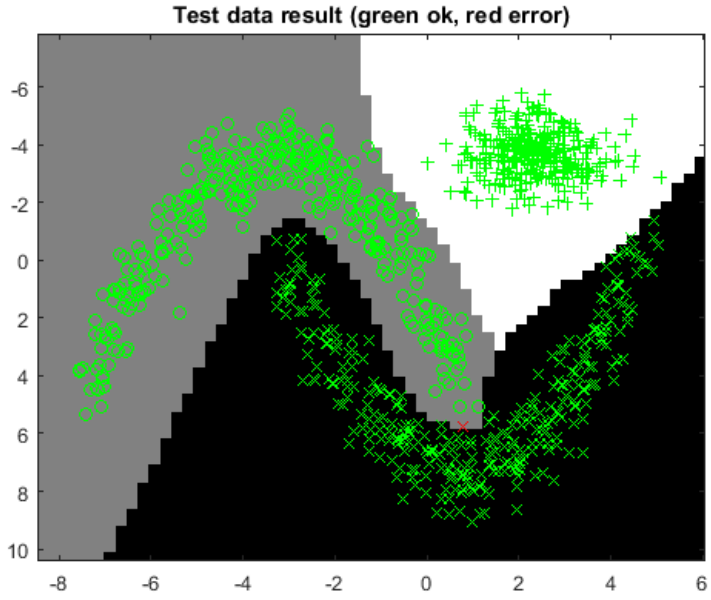

Figure 7: Learned boundary of the third data set

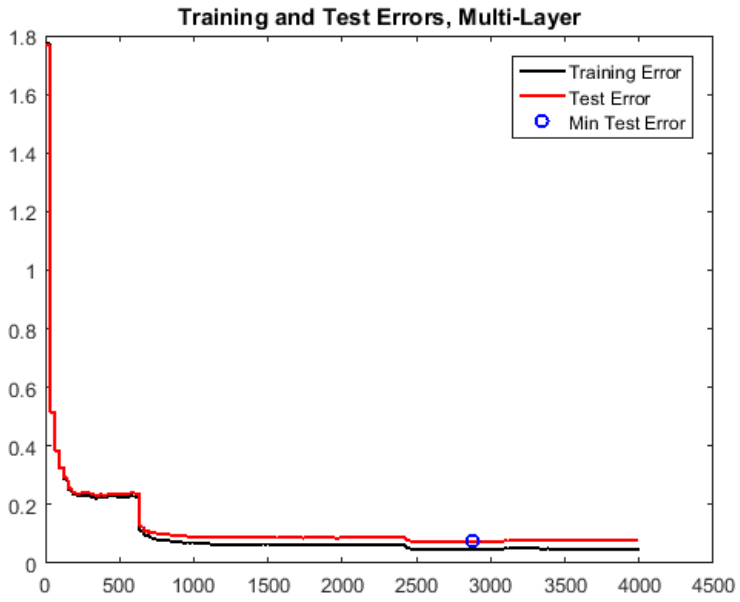

Figure 8: Error of the fourth data set

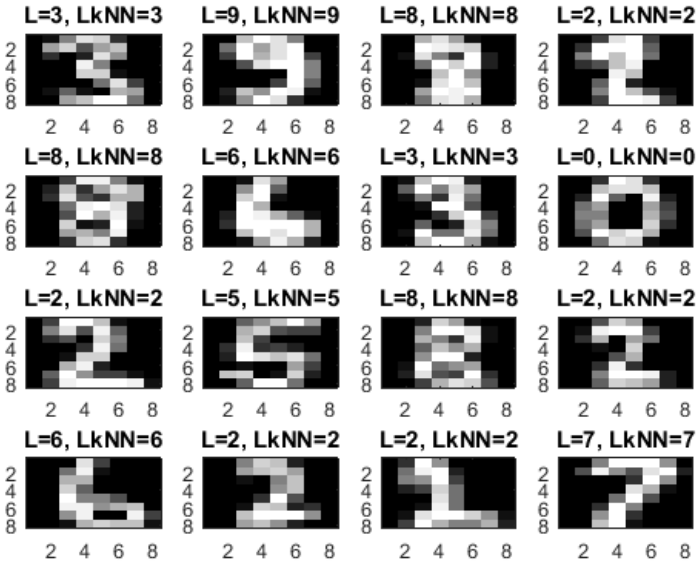

Figure 9: Learned boundary of the fourth data set

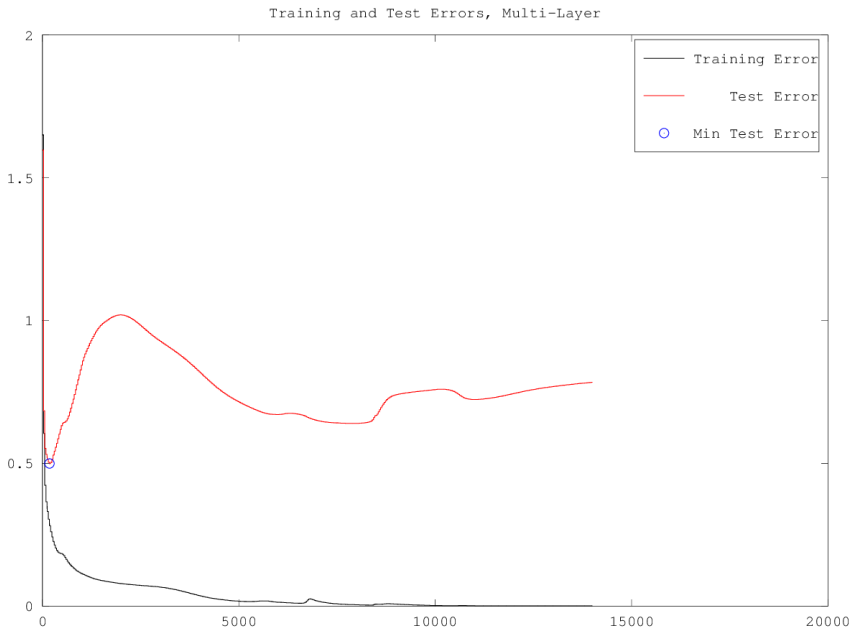

Figure 10: Error curve in overfit scenario

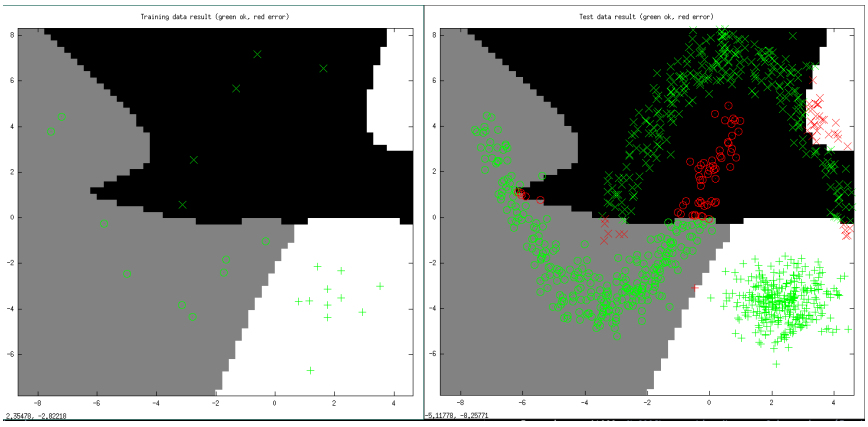

Figure 11: Overfit boundary