
Machine Learning (CS 391L) : Spring 2013
Homework 1

Vineet Keshari (EID vk3226)
vkeshari@cs.utexas.edu

Files in submission

1. hw1.m : The main script.
2. hw1FindEigenDigits.m : Computes mean vector & eigenvectors.
3. hw1KNN.m : Classifies based on K-nearest neighbors.
4. hw1LAv.m : Classifies based on class means.
5. hw1_vis.m : Visualization utility

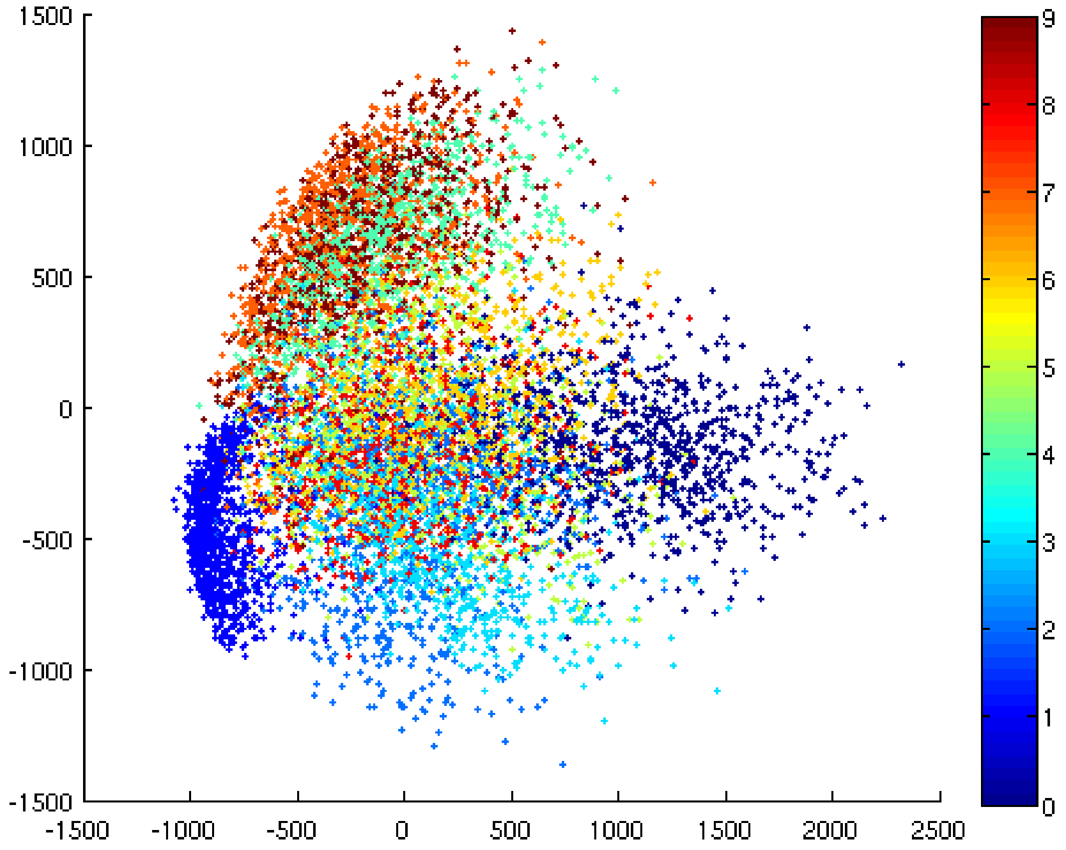
Approach

The main script loads training and test examples randomly from *trainImages* and *testImages* respectively. Images are vectorized column-wise. The eigenvectors are then calculated using the function *hw1FindEigenDigits*, followed by a projection of both training and test images to the resulting basis. The technique used to do this is as described in the notes available on the course website. Once we have the vectors in reduced space, we classify them based on either k-nearest neighbors or per-class mean vectors (the latter approach is as described on page 21 of notes). To avoid a disproportionate training set due to random sampling from train and test images, the entire process is repeated *noOfIterations* times and the classification results are averaged.
Note: More details about the code can be found as comments.

What does the data look like?

If we reduce the no. of dimensions to 2 (by taking eigenvectors corresponding to only the two largest eigenvalues), we can visualize the data on a 2-d scatter plot. The result for 10,000 training images is shown in the figure below. We can make the following observations:

- 0 and 1 are almost perfectly separated.
- There is high overlap between {3,8}, {4,7,9} and {2,3,5}.



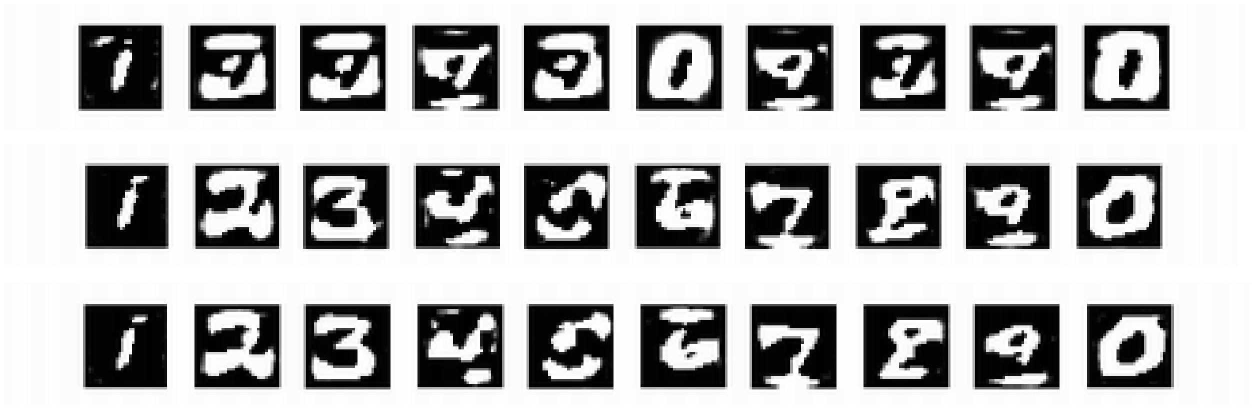
In particular, 2 and 5 appear to inhabit the same region centered around the origin, and do not form clusters like the others. However, this does not mean that the two classes are inseparable in n-dimensions as well. The purpose of visualizing the data in 2-d is just to get a good sense of how the data is distributed in space.

Mean class vectors

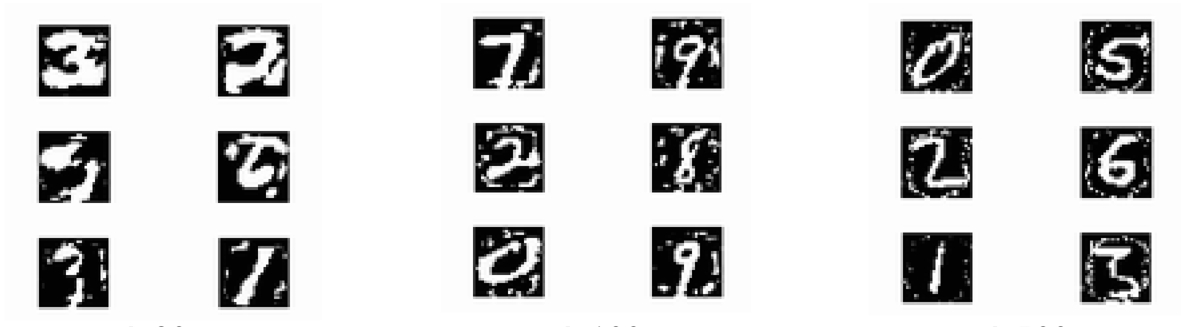
A good way to get an intuition about the no. of dimensions required for good classification is to look at the mean vectors reconstructed after projection to reduced basis. Below are the vectors regenerated from (top to bottom) 2, 20 and 100 dimension basis.

We can see that, as noted in the previous section, {2,3,5} and {4,7,9} are indeed indistinguishable in 2 dimensions, and it is clear that we need more dimensions to correctly classify the digits.

The mean vectors are much more distinct for 20 dimensions, while they are visually unique for 100 dimensions.



Information Loss
Next, we look at the information loss in individual training or test vectors when we follow the same process. We do this for individual examples since k-means classification is done based on individual training vectors instead of the mean vectors, so our above analysis wasn't exhaustive. Below is a reconstruction of some training digits after projection to (from left to right) 20, 100 and 500-dimensions.

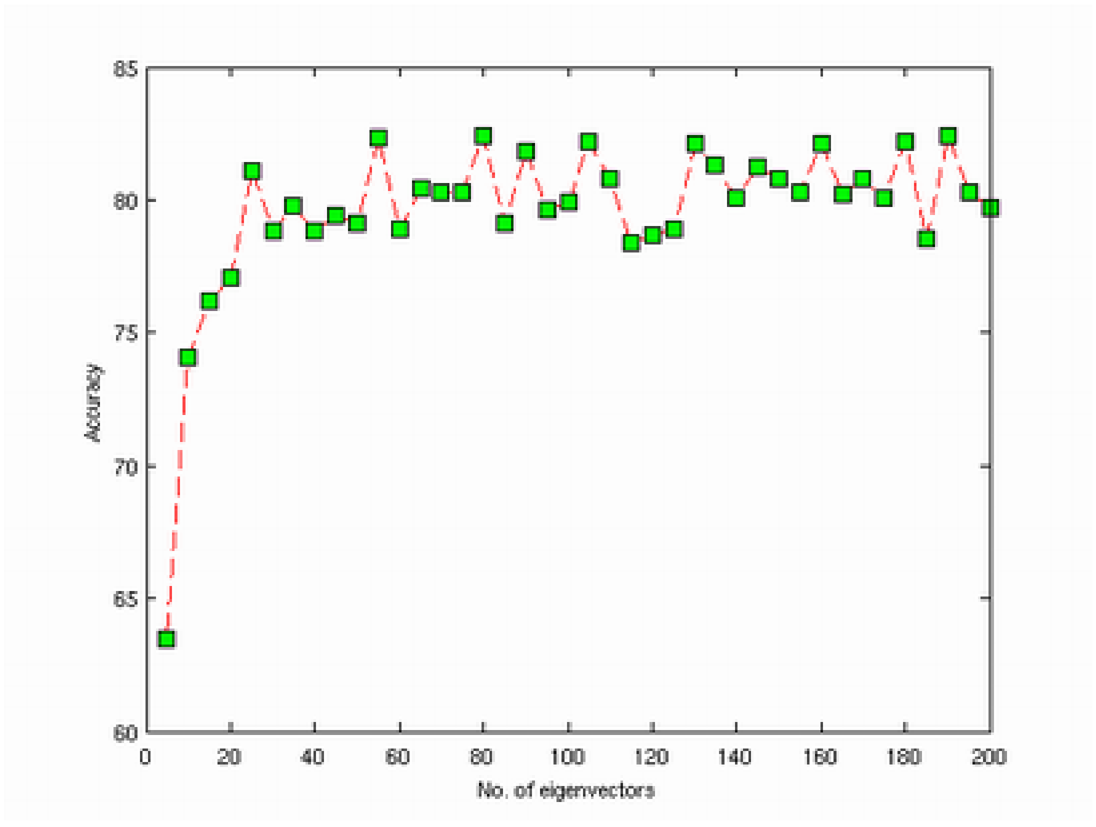


d=20 **d=100** **d=500**

We can see that for d=100 and above, we get visually distinct reconstructions, while at d=20 the images are not clear but may still be good enough for classification.

In either case, it is clear from the above analyses that between 20 and 100 dimensions should be enough for our classification.

So how many dimensions are good enough?
We will now empirically determine this by gradually increasing the no. of dimensions on training and test data. The graph below shows the results for k-nearest neighbor classification (k=4), using a training set of size 100, and tested on 100 vectors. As stated earlier, results are averaged over multiple random sets.

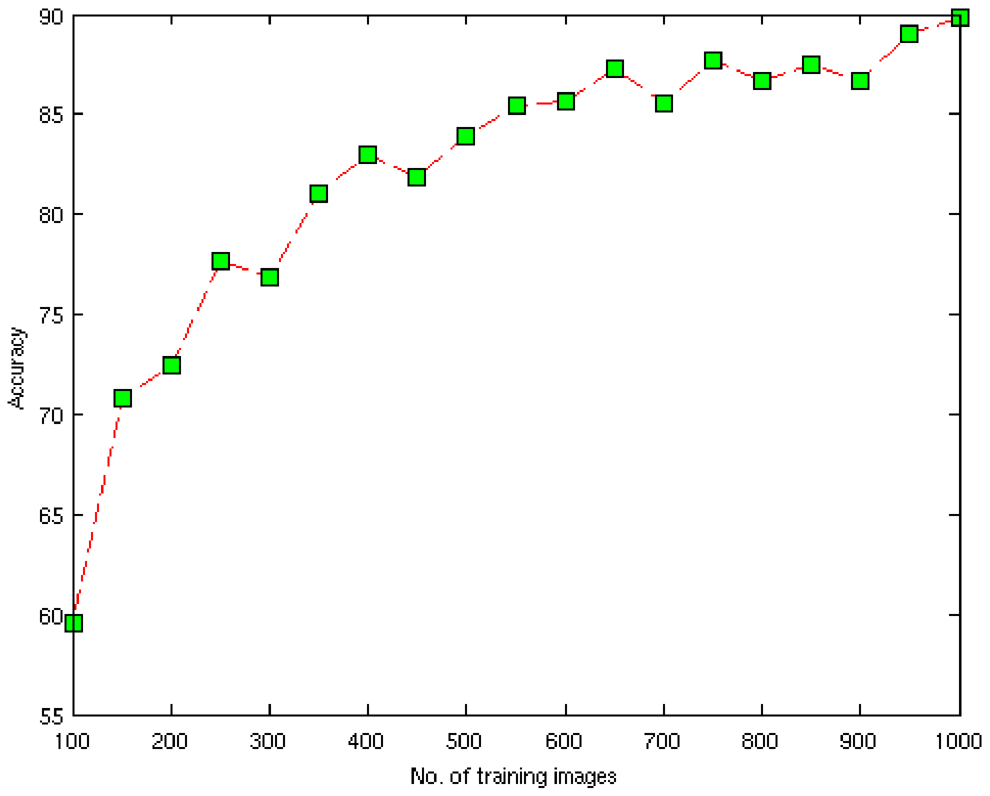


We can see that the accuracy increases up to about 30 dimensions and then levels out. Note that the accuracy is in the low range [65,85] since I ran this for only 100 training images (the algorithm is very slow for large training sets).

How many training examples are good enough?

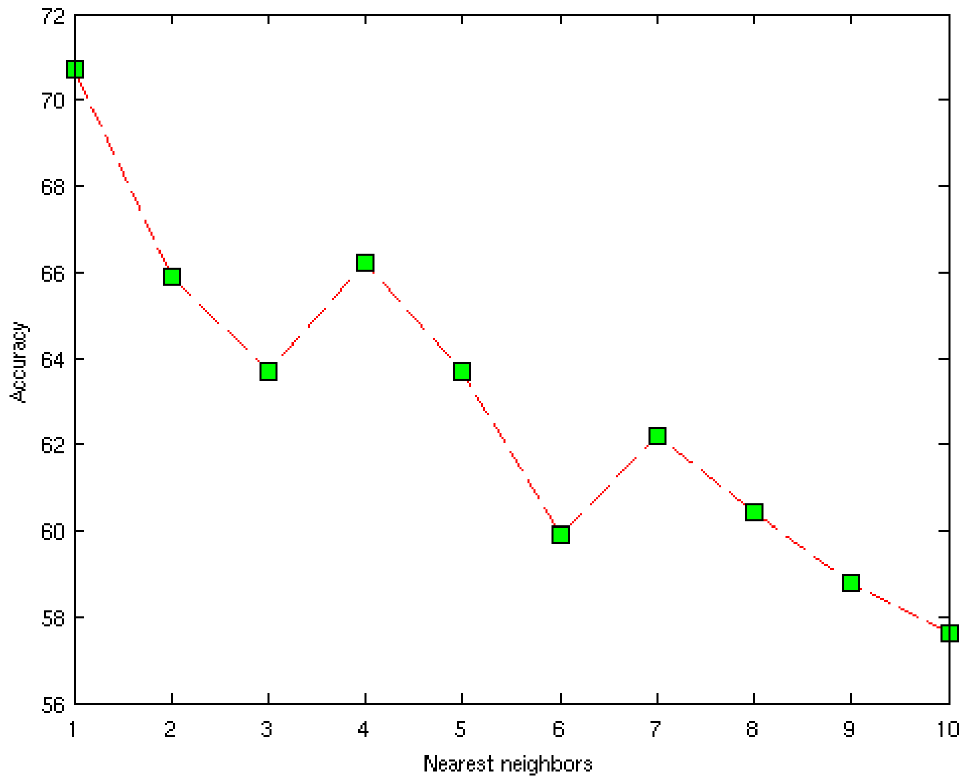
We now look at how the accuracy changes as we increase the no. of training images. The parameters are the same as in the previous experiment and the no. of eigenvectors is 100. The results are shown in the graph below.

It can be seen that the accuracy increases to 90% for a training set of 1000 and continues to rise beyond that. I couldn't run the algorithm for intermediate values as it would be very slow, but I got an accuracy of 95.16% with a training set of 10,000. This shows that the accuracy increases much slower for larger training sets than the range [100, 1000] shown in the graph, indicating that it tends to level out at ~95% at some point between 1000 and 10,000.

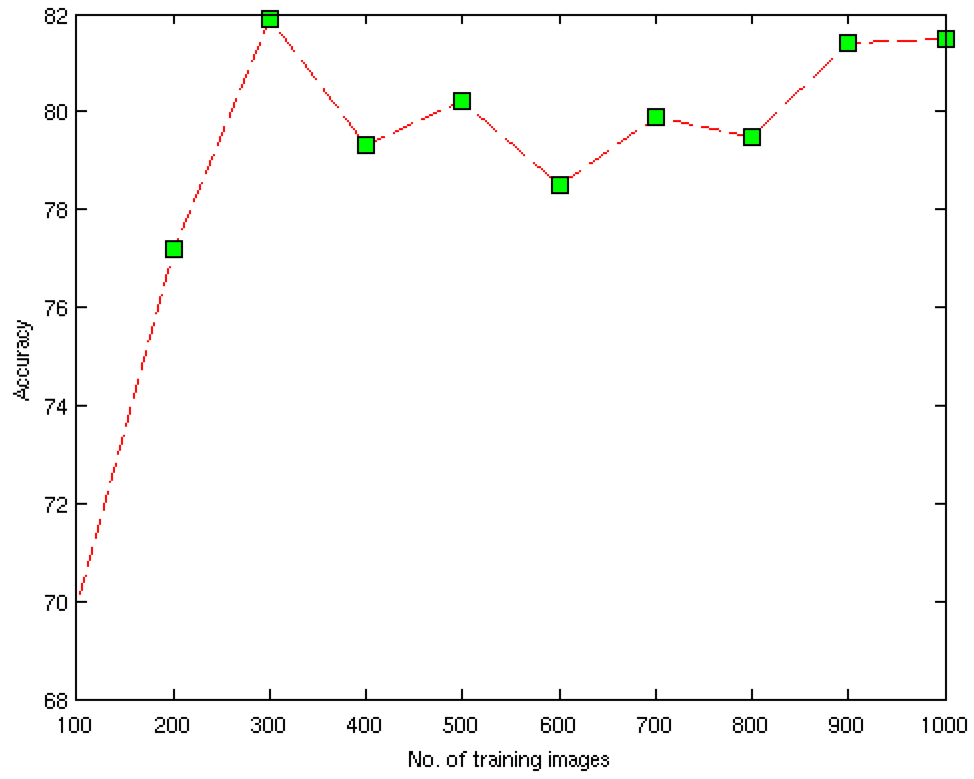


No. of nearest neighbors (k)

The graph below shows the classification accuracy as the value of k increases. Using the same setup as before, we see that the accuracy decreases as k increases. This behavior is counter-intuitive, and I don't have a good explanation for it. It should be a valid assumption, though, that some classes overlap even in higher dimensions and so the resulting boundaries between classes may have several misclassified points. Another possible explanation is that because only 100 training examples were used, there were only ~10 samples of each digit, causing less reliable classification near the class boundaries, specially for large values of k.



72
70
68
66
64
62
60
58
56
Accuracy
1
2
3
4
5
6
7
8
9
10
Nearest neighbors



82
80
78
76
74
72
70
68
Accuracy
100
200
300
400
500
600
700
800
900
1000
No. of training images

k-nearest neighbors vs class means
Finally, we look at how the k-nearest neighbors classification performed in comparison to classification based on nearest class mean. The graph for the latter method for various sizes of training sets is shown above, and it is clear that the accuracy levels off at 80% after initially increasing. Thus, k-nearest neighbors (with 90% accuracy for 1000 training examples) is empirically more accurate.

Conclusion
Based on our observations, we note the following:

- About 30 eigenvectors are enough to classify the digits with good results.
- Classification accuracy increases with increasing number of training images and can be at least as high as 95.12%.
- k-nearest neighbor classification is the more accurate classification method.
- The accuracy of k-nearest neighbors seems to decrease with increasing values of k, with k=4 giving better results than most others.

Further Work

- I think a good next step for this experiment is to try a different classification technique such as logistic regression on this data.
- For input processing, it should help to think of an approach different from linearizing the pixel intensities row-wise or column-wise. Since most images have pixels in the center, the vectorized images mostly look the same. Further, properties such as relative positions and orientations as well as line thickness vary greatly for different samples of the same digit.
- Instead of hyperplanar classifiers, we can use curved-surface-like classifiers such as Principal Manifolds to get better accuracy.