Homework 2: Prinicipal Components Analysis

Selected from student's homework

February 16, 2012

1 GOOD REPORTS 1

1.1 Introduction

This assignment focuses on using Principal Components Analysis (PCA), a method of identifying the vectors describing the greatest variance within a dataset. This vector space can be used as a means of classifying other vectors in the original space according to similarity. A naive approach would simply take the eigenvectors of a dataset's covariance matrix to accomplish this, however in this paper it will be shown that not only can a subset of these eigenvectors be used, but the subset can be obtained from a lower-dimensional space to improve computational performance, with minimal sacrifice to accuracy.

1.2 Implementation

The PCA algorithm was implemented using Matlab and was run on a provided set of training and test data. Labels for both datasets were provided for analysis purposes. The number of training samples used varied with experiment and generally stayed under 784, the number of pixels in each image. Unless otherwise noted, using k data points implies that the data points are the first k in the provided set.

The general approach to reducing dimensionality assumes that the training set size far exceeds the dimensionality of the individual samples, however in this case the opposite is true. Nevertheless, the equations for this analysis were formulated in such a way as to reduce the dimensionality of the samples. Thus the training set sizes were under the count of 784 pixels that represent each image.

The algorithm used for classification is as follows. Prime notation ($\prime$) is used to indicate that a variable is in the reduced eigenspace.

Algorithm 1.1

1. Take the mean of the training set: μ
2. Subtract the mean from each vector in the observation set A to get the converted observation set: $I = A - m$
3. Find the resulting covariance matrix: $\Sigma = I^T \cdot I$.
4. Get the top n eigenvectors of Σ, ordered by eigenvalue: V'
5. Project these vectors into the original space to obtain $V = \{u_i\}$
6. Normalize V such that $||v_i|| = 1$
7. For each test item t, project t into the eigenspace: $t' = V^T \cdot (t - \mu)$

8. Similarly, project each $O \in I$ into the eigenspace and use a Euclidean nearest-neighbor algorithm to identify the closest match $\Omega \in I'$
9. Apply the label of the corresponding $O \in I$ to t.

By comparing the applied label to the supplied label it is possible to determine the accuracy of this classification algorithm using a variety of parameter configurations. Section 1.3 will describe these in detail.

1.3 Experiments

1.3.1 Reconstructions

Reconstructions were performed as follows:

1. Given an observation O, take its projection $\Omega = O'$ as described in Algorithm 1.1
2. Multiply each u_i by its corresponding ω_i, the entries of Ω, and add these together. Equivalently, obtain $r = V * \Omega$
3. r is now the reconstruction of O

Figures 1 and 2 show the improvement when increasing the number of samples from 28 to 784. The first samples are fairly blurry and show that a larger eigenvector set is needed to produce images that are visually similar to the originals.

1.4 Eigenvector Visualizations

Similar to Experiment 1.3.1, the eigenimages were "reconstructed" from the set of eigenvectors V. Because the eigenvectors are normalized, we multiply each by the scalar 255 to obtain 8-bit image vectors. The results of these reconstructions are shown in Figure 3.

1.5 Classifications

The objective of this assignment was to build a system capable of classifying the provided test images, so these experiments focus on the effectiveness of the algorithm in accomplishing that goal. The first question to answer was whether a large training set was necessary to achieve a high degree of accuracy.

Figure 4 shows the classification accuracy on the first 5,000 test images across varying numbers of training samples. In each case, the maximal number of eigenvectors was used, so the eigenvector count is equal to the training set size. Figure 5 shows similar results on the last 5,000 test images, which were less clear and presumably more difficult to classify than the first.

The data show that, as we would expect, increasing the number of training samples significantly increases classification accuracy for low set sizes. As sizes exceed 300 the gains drop off quickly, approximating the shape of a logarithmic curve.

Surprisingly, accuracy began to stabilize at almost 10% higher in the complex case than in the simple case. It is unclear precisely why, yet this highlights a classic misconception with computer vision problems. A classification task that is simple for humans need not be simple for a particular algorithm, and vice versa.

1.6 K-Nearest Neighbors

In a Euclidean nearest-neighbor algorithm it can be beneficial in some circumstances to look at the k nearest neighbors. The experiments thus far have used the k-nearest neighbors algorithm with $k = 1$, so this experiment analyzes the effects of increasing k, up to a maximum of 15.

Figures 6 and 7 show the results of this experiment. Similar to Experiment 1.5, tests were performed on the 5,000 simple and complex test images, although this experiment only used 784 training images. The data show that increasing k has almost no beneficial effects on accuracy for small k, and shows some detrimental effects as k increases. This should be expected for large values of k, since as $k \to \infty$, the classification tends toward the mode of the entire dataset. The result is surprising for small values, although easily explainable as an inherent property of the given dataset.

1.7 Eigenvector Counts

It is possible to choose the top n eigenvectors, in descending order of eigenvalue, and still obtain accurate classification results. Figure 8 shows that after the first 15 eigenvecors the increase in classification accuracy becomes insignificant. This is somewhat peculiar when compared with the results of Experiment 1.3.1, since 28 eigenvectors was not sufficient to create visually pleasing reconstructions. As mentioned in Experiment 1.5, there need not necessarily be a connection between the effectiveness of human classification and algorithmic classification.

1.8 Implementation Issues

This implementation was particularly difficult to debug. The dimensionality of the sample images inhibited any kind of manual evaluation of intermediate procedures, so results had to be evaluated as a whole. In particular, a bug in the code was causing reconstructed images to appear distorted. The bug was ultimately identified as an error in the normalization process.
After correcting the error, the overall effectiveness of the classification algorithm and the visual accuracy of the reconstructions were sufficient to suggest that the algorithm was correct.

1.9 Conclusion

The experiments showed that this PCA implementation was able to classify images with a high degree of accuracy given even a modest training set. This success was mirrored by the reconstruction process. Overall the implementation achieved the goals of accurate classification and reconstruction.

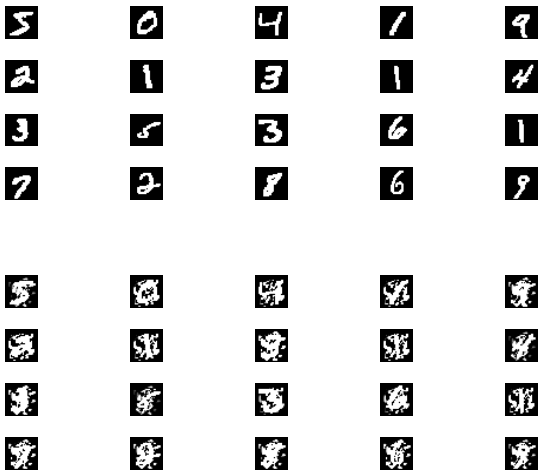


Figure 1: Images reconstructed from 28 trial images and eigenvectors. The 25 images on the top are from the training set, and the corresponding images on the bottom are reconstructed as described in Experiment 1.3.1.

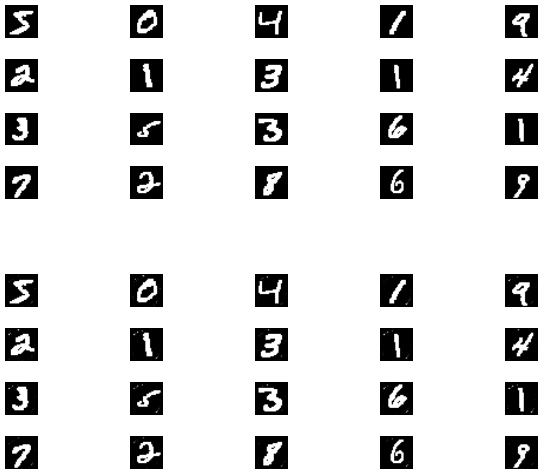


Figure 2: Images reconstructed from 784 trial images and eigenvectors. The 25 images on the top are from the training set, and the corresponding images on the bottom are reconstructed as described in Experiment 1.3.1.

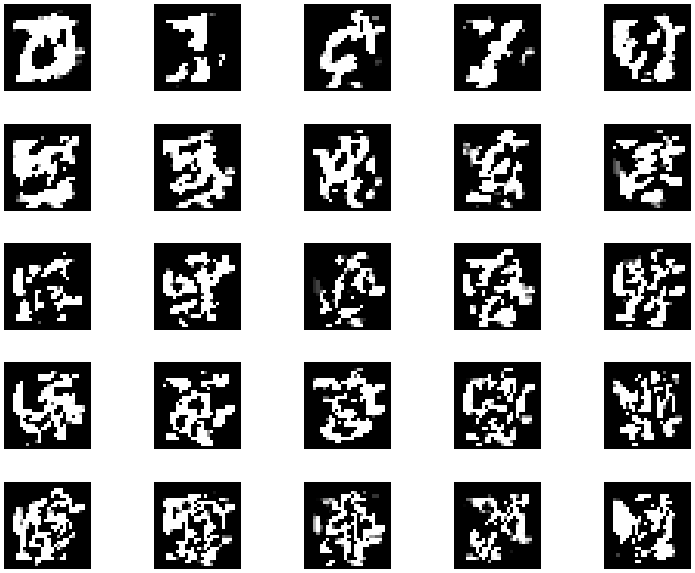


Figure 3: Reconstructions of the 25 eigenvectors obtained from a sample of the first 25 training images.

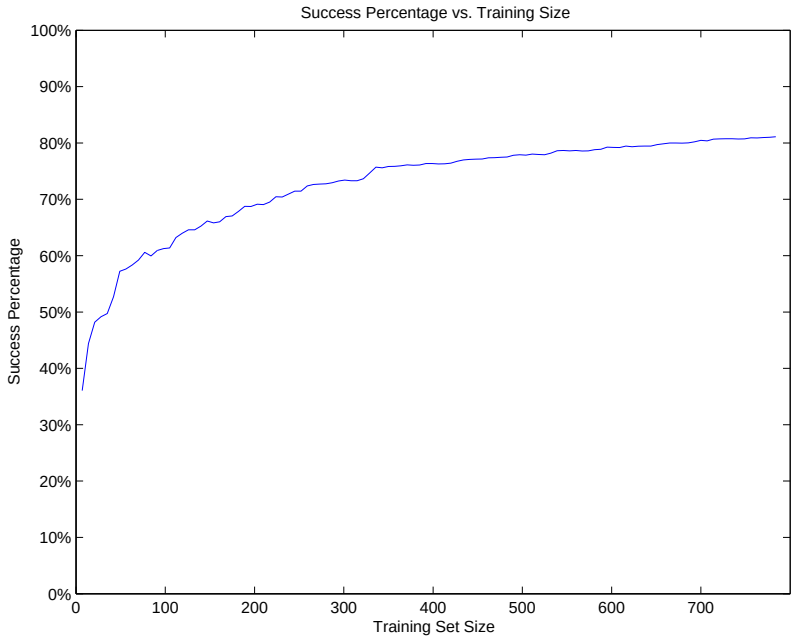


Figure 4: A plot of successful labeling percentages for the first 5,000 test images versus the size of the training set used for determining eigenvectors. These test images represent the simple set.

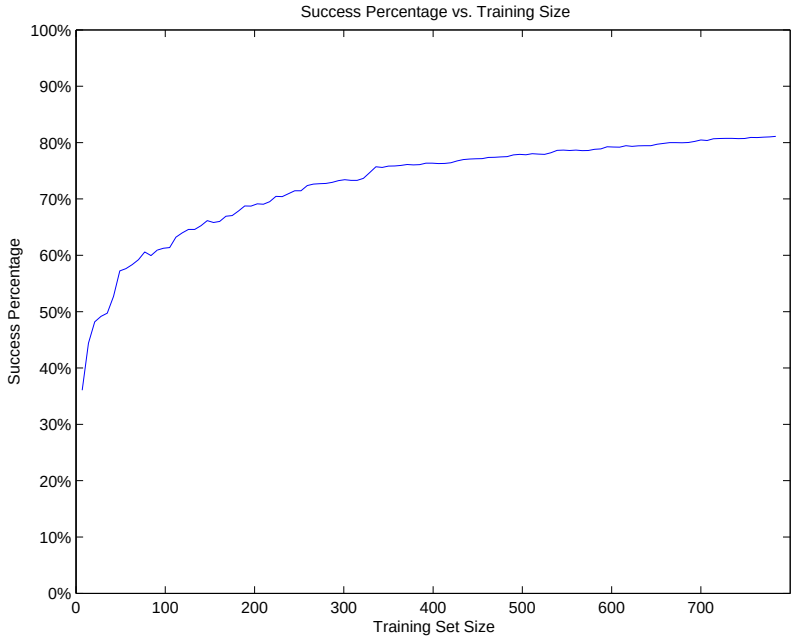


Figure 5: A plot of successful labeling percentages for the last 5,000 test images versus the size of the training set used for determining eigenvectors. These test images represent the complex set.

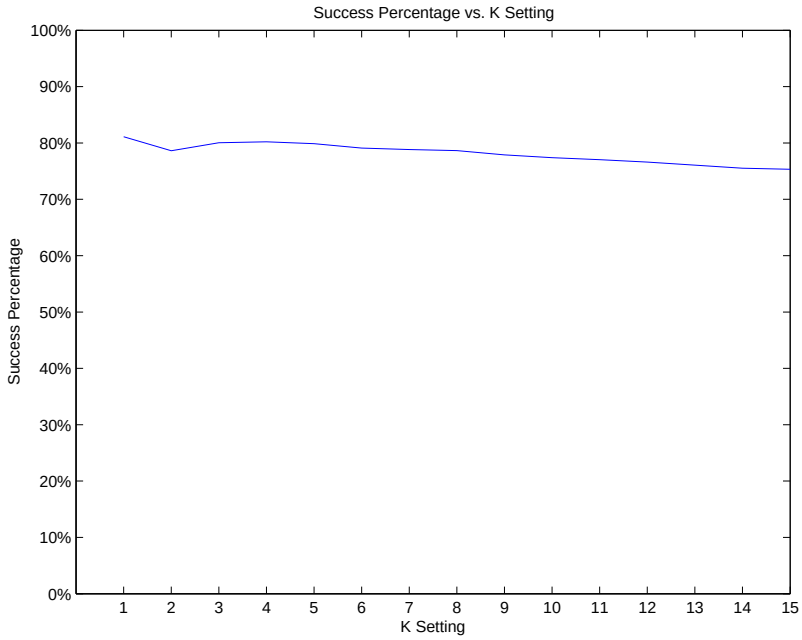


Figure 6: A plot of successful labeling percentages for the first 5,000 test images versus the k setting in the k-nearest neighbors algorithm. These test images represent the simple set.

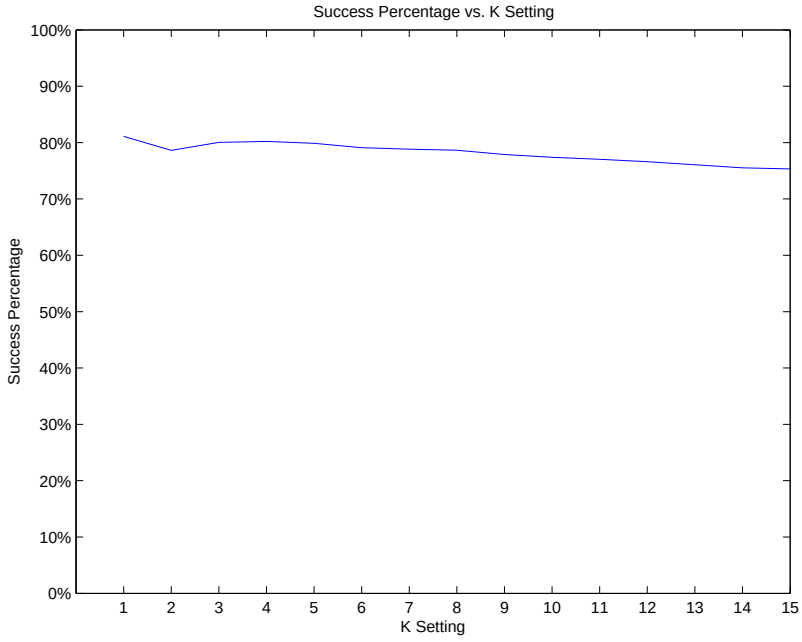


Figure 7: A plot of successful labeling percentages for the last 5,000 test images versus the k setting in the k-nearest neighbors algorithm. These test images represent the complex set.

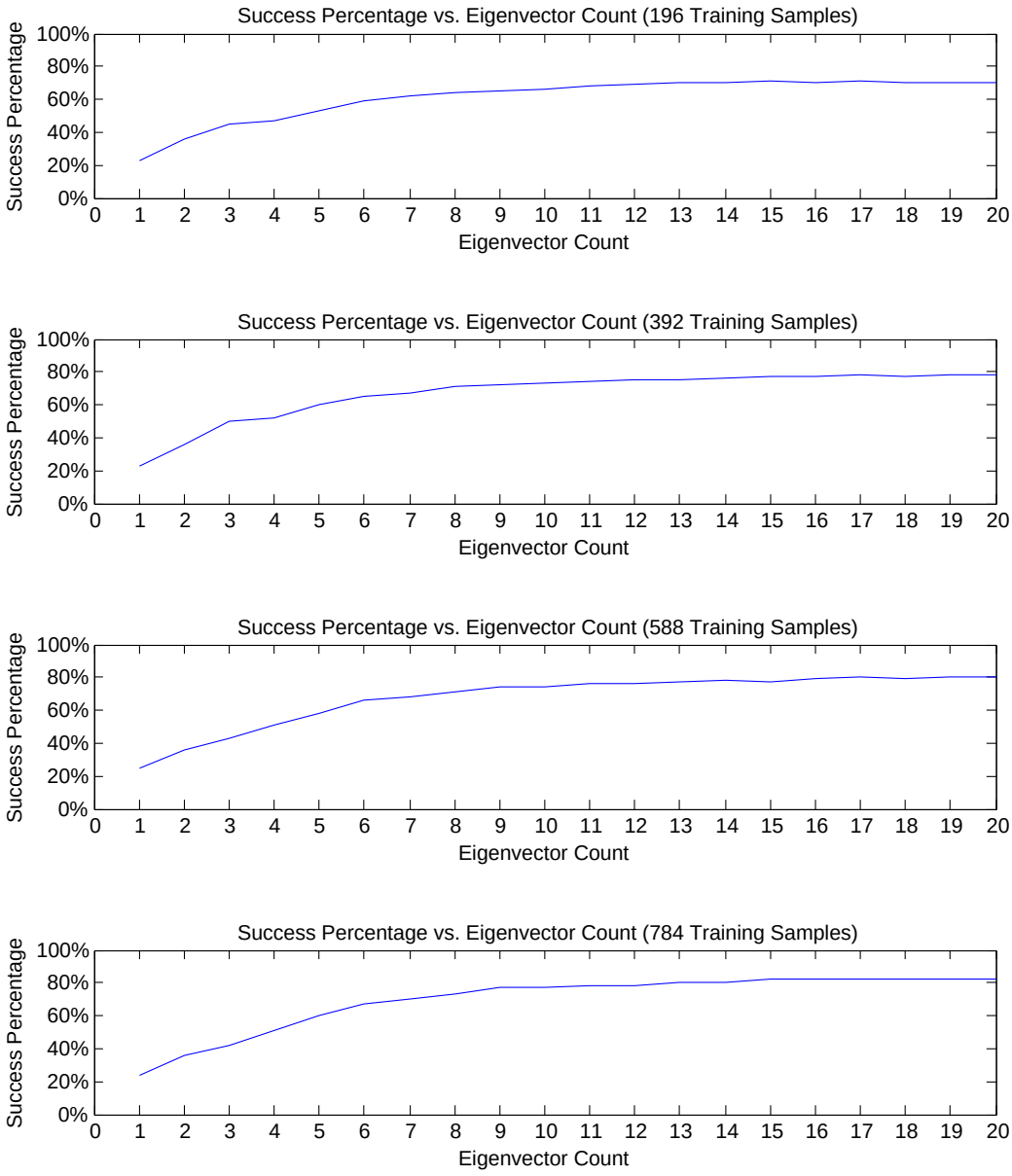


Figure 8: Plots of successful labeling percentage vs. the number of top eigenvectors used.

2 GOOD REPORTS 2 - Reconstruction Display

One good experiment

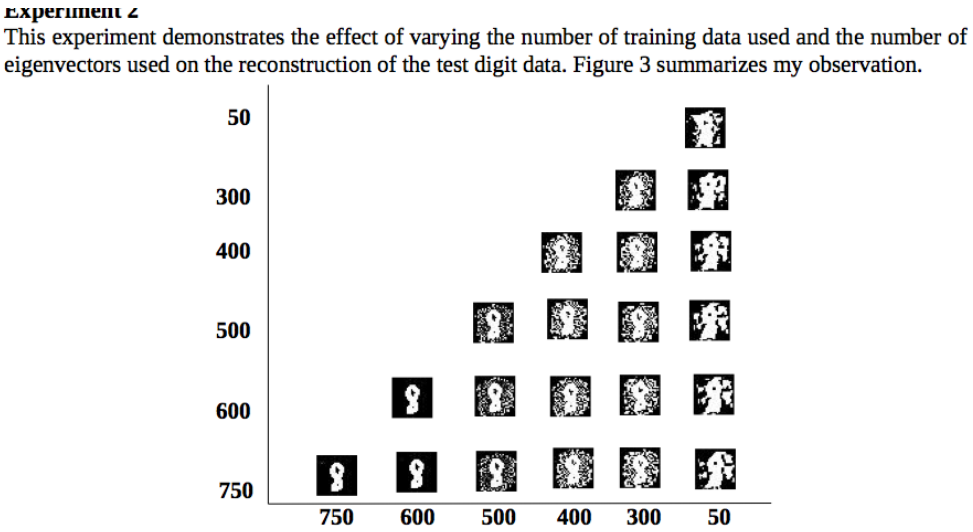


Figure 3. Impact of varying the # training images and # eigenvectors used for reconstruction

3 GOOD REPORT 3 - Accuracy Plot

3.1 Classification Accuracy

To obtain the accuracy quantitatively, I set up experiments of digit image classification for different number of training images and different number of eigenvectors. In this experiments, the number of training images was chosen from $\{50, 100, 300, 500\}$, and the number of eigenvectors was chosen from $\{1, 2, 5, 10, 20, 50, 100\}$. For the K-nearest neighbor algorithm, I chose $K = 10$, which is the same as in the previous experiment.

Figure 9 shows the experiment results. If 500 training images are used, the algorithm can classify test digit images accurately up to 80%. Clearly we see that the classification accuracy increases as the number of training images increases. For example, if we use 300 training images, the classification accuracy is up to 75%, and it gets worse if we use fewer training images. However, it is not always true that the accuracy increases as the number of eigenvectors increases. When we increase the number of eigenvectors up to 10, clearly the classification accuracy becomes higher, but it does not get higher at all if the number of eigenvectors increases more than 10. It looks like that the accuracy meets a limit where more eigenvectors do not result in higher accuracy.

3.2 Discussion

It is not intuitive that if we use more eigenvectors then the reconstructed image looks more like the original image but the classification accuracy does not always increase. Comparing the accuracy results for 100 training images in Figure 9 and the reconstructed images in Figure **??**, we can see this fact more clearly.

This result shows that closer images in the eigenspace does not mean that they look more like each other. As a future work, we can look into the image points on the eigenspace to find how they are

clustered. Also, it will be a good problem to find how we can exploit visual similarity of training images to find principal components of them.

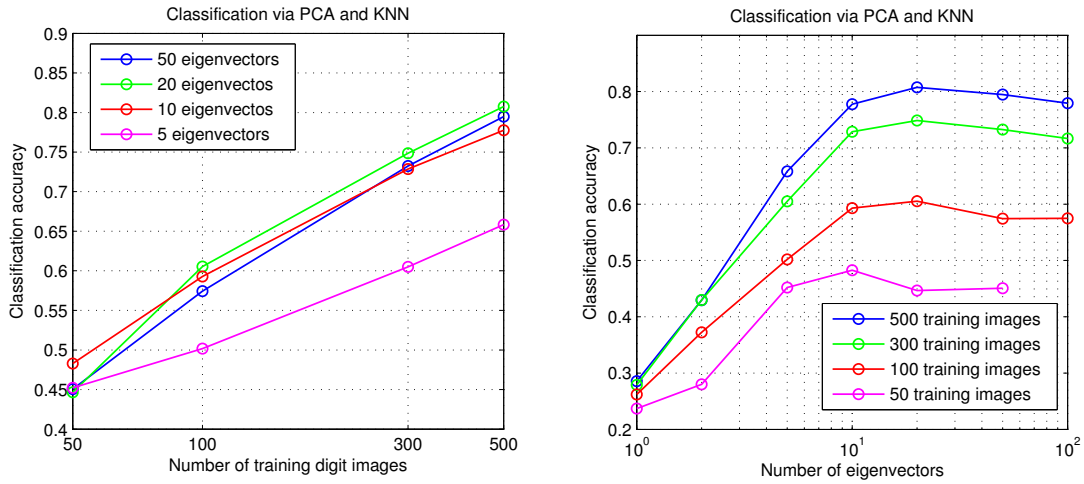


Figure 9: Digit classification accuracy over the number of training digits and the number of eigenvectors (the dimension of eigenspace), respectively.

4 GOOD REPORT4 - Factor Analysis

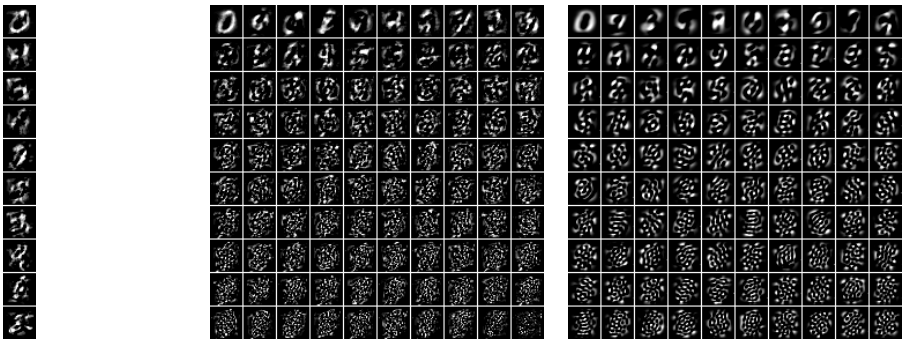
4.1 Algorithm

The PCA (Principal Component Analysis) algorithm is pretty straightforward, and the MATLAB implementation closely follows the general instructions. My `hw2FindEigendigits` function is a short 12 lines of code, and could be much shorter by embedding many of the functions. One complication (due to the original version of the assignment) was that sometimes the width of the matrix did not exceed the number of original dimensions of the data (784 in our case), and since the function was required to return a matrix the same size as the input image data, I found that padding it out with zeros was a suitable solution. In those cases where the number of training images was less than 748, truncating the eigenvector matrix to match that number was easy enough. Also, the assignment noted that the eigenvector matrix would need to be sorted by eigenvalues, but MATLAB's `eig` does that automatically, and only a reverse is needed. However, in line with the assignment, I still sort the eigenvalues and return the corresponding eigenvectors. I have produced a sampling of eigenvectors in Figure 10 for different amounts of training data. The first few eigenvectors of each group seem intuitively reasonable, vaguely resembling the '0' shape that covers much of the space used by a digit. However, after the first ten or so, they seem to become random and hardly distinguishable.

4.2 Evaluation

I tried out four basic evaluation metrics:

- kNN (k = 1) (kNN: k-Nearest Neighbors)
- kNN (k = 10) – majority vote



(a) Using 25 training images **(b)** Using 100 training images **(c)** Using all training data (60,000 images)

Figure 10: The first 10 or 100 eigenvectors (ordered by rows, not columns)

- kNN (k = 10) – weighted vote
- Cosine similarity

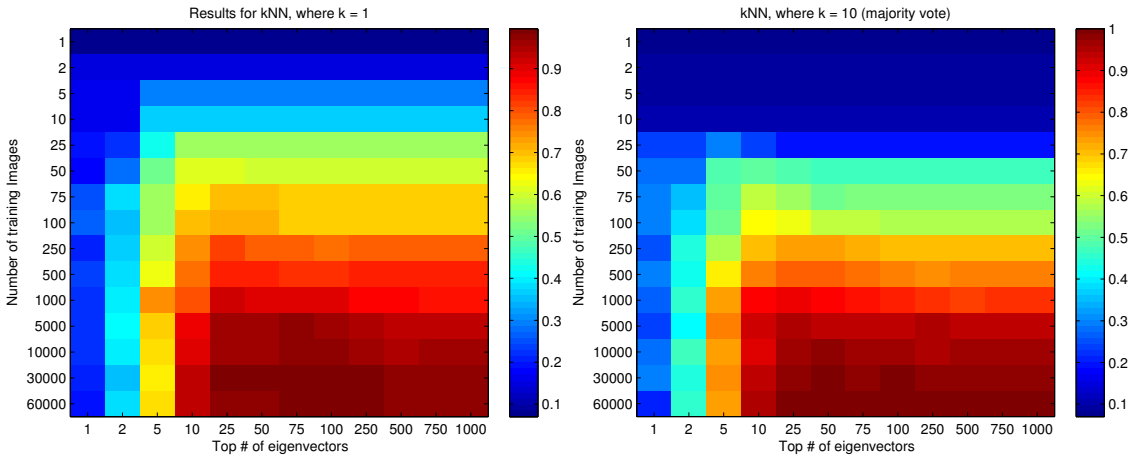
Most of the computation time is spent evaluating each point relative to the training data. It's easy to project all the training/labeled data and the test data into the space of reduced dimensionality quickly, so I only had to do that once per choice of evaluation metric, number of training images, and number of top eigenvectors. But to evaluate a large group of test data, the algorithm would evaluate each one in term, analyzing the entirety of the training data to find the closest points. If speed was an issue, we could probably do some preprocessing on the training data to assign different regions of the PCA space certain labels, and then the test points could be queried against these regions, instead of all 60,000 training points.
At first, I used kNN with k = 10 and took the most popular label from the resulting list. This ended up being slightly better than kNN with k = 1, since in some cases you can have a particularly sloppy '4' that looks a *lot* like some other specific and also very sloppy '9,' but in general looks more like a '4' than a '9,' based on several other digits. That is, it takes out some of the anomalies. The advantages of this diminishes as we gain more training images, as the PCA space becomes more saturated with examples. But then I realized that this was naive, and that since the `knnsearch` function returns distances as well as a simple indexation, I could average the distance to all the '4' matches in the top 10, and all the '9' matches, for example, and take the shortest of those. This turned out not to really make a difference, but since it seemed to make more sense, I proceeded to use this metric (weighted kNN, k = 10) for the final analysis.
Cosine similarity (using `pdist2`, not the cosine option of `knnsearch`), gave me decent results, but it wasn't nearly as fast or successful as `knnsearch` using euclidean distances. Also, MATLAB warned me that my data did not seem entirely suitable for the cosine similarity metric, which is probably true.
See Figure 11 for a comparison of these different metrics over a limited number of test images. (I only used 200 images in these plots, since evaluating 10,000 takes quite a bit of time.)

4.3 Results

There are two variables we can plot accuracy on: the number of top eigenvectors (the dimensionality of the eigenspace), and the number of training images. I do this for different portions of the test data:

We see that we can get pretty good results for as few as 500 training images while using just the top 50 eigenvectors. Perhaps the most interesting thing about these graphs is that for fewer training images, in the 100-10,000 range, it helps to truncate the eigenvectors, 'de-noising' the data when transforming it into the PCA space, in a way. However, this advantage falls off as we simply use more and more training images (which is just another way of reducing noise in the training dataset).
Overall, we achieved up to 93.28% accuracy on the "easy" images (using the full test set, and the top 50 eigenvectors), and 97.18% on the "hard" images. I didn't visually compare the different sets of the last 5,000 and first 5,000 of the 10,000 training images, so I must have accidentally put the last 5,000 results into the 'easy' results. That or the assignment got the ordering backwards, because one would expect lower accuracy for the "harder" images. The top accuracy for the full 10,000, measured in a separate run, is predictably the mean of the other top accuracies: 95.23%.
The most interesting observation was that less eigenvectors give better results, but less notably so with more training images. But another interesting problem was the matrix manipulation to average the distances from `knnsearch` while grouping by label. This would be called 1,800,000 times for the full evaluation of the 10,000 test images alone, so I wanted to avoid a for-loop if possible. So I created a matrix of mostly zeros from the list of labels returned by `knnsearch`, which, when multiplied by the distances, would sum the distances for each label, returning a vector as long as the number of labels. Then I just divided by the vector-sum of that multipler matrix, and had my averages.
NOTE: Figures on next 2 pages.

Acknowledgement: Thanks for the contribution of some students :) If you have any questions, please email us. Thank you.



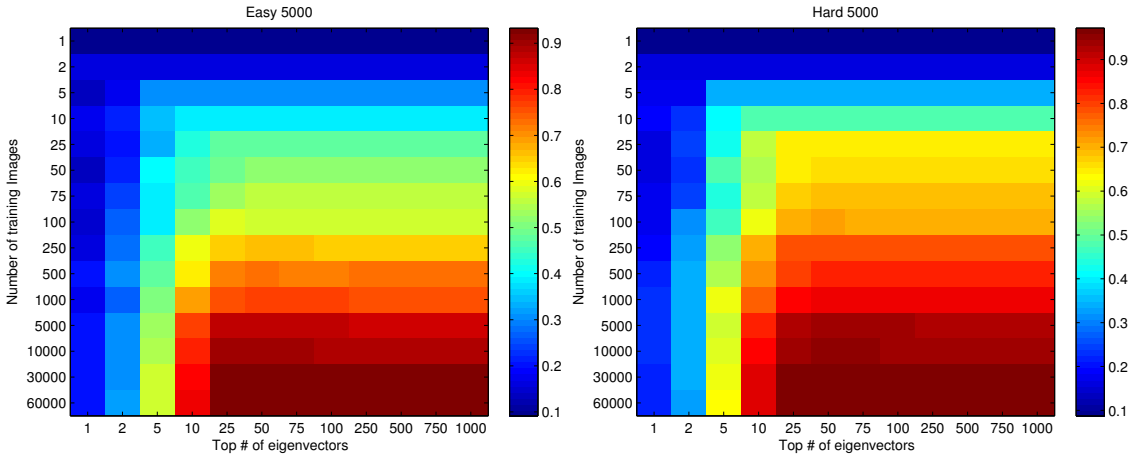
(a) 1-NN

(b) 10-NN majority vote

(c) 10-NN weighted

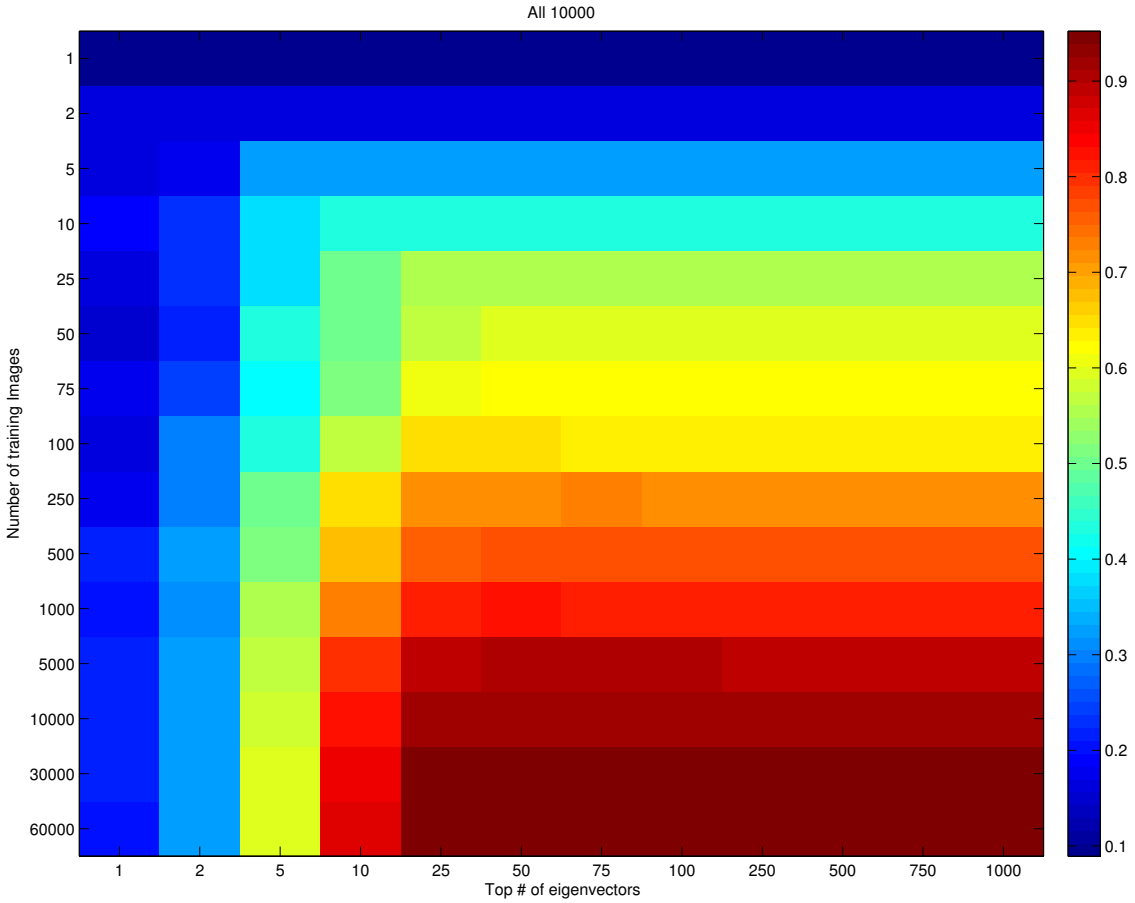
(d) Cosine similarity

Figure 11: Evaluation metrics. The hotter the color, the more accurate the combination of top eigenvectors and number of test images. Please note that these may look blurry due to the non-standard way Mac OS X Preview renders small images. Use Adobe Acrobat to see a correct rendering.



(a) Easy 5,000

(b) Hard 5,000



(c) All 10,000

Figure 12: Results