CS 391L Homework1 Report
Haocheng An (UTEID HA8886)

I Introduction

Principle Component Analysis (PCA) is an important tool in machine learning. In many times, it is used in classification problems. The idea behind PCA is to find the major component and classify the unknown data to one of the types based on the similarity. Also, by conducting linear transformations on given vectors, I can reduce the dimensions of data I use and thus make the classification easier. In this lab, I am facing 60,000 training images and 10,000 test images. All these images are of size 28*28. Each image represents an eigendigit—a picture reflects the digit from 0 to 9. By summarizing the key features in training images, I use them as a base for new predictions. The 10,000 testing images providing great tools for examining the features I got. PCA is a great fit for this problem as different representations of same digits occupies would occupy similar small pixels and the training set and the test set are both coming from same 10 categories. As 28*28 = 784 is a relatively big dimension, by using PCA, the dimensions can be reduced as well.

II Methodology

First, I load in the training data and testing data inside the python program. For the training data, I calculate the mean column Vector. After getting the mean vector, I subtract it from every column vector as that is the major component and denote the matrix after subtraction as V. After getting rid of the major same component, I try to work on the specific feature component. This is achieved by finding eigenvalues and eigenvectors of V. As every picture is of size 28*28, which means every picture lives in the Euclidean space of R^{784}.

To solve a polynomial equation of order 784 will take up much computation resource. But by the definition of eigenvalues and eigenvectors, I have following facts that always hold:

*By Convention:

Upper case letters are used to denote Matrices.

Lower case letters are used to denote Vectors.

Greek letters are used to denote scalars or constant.

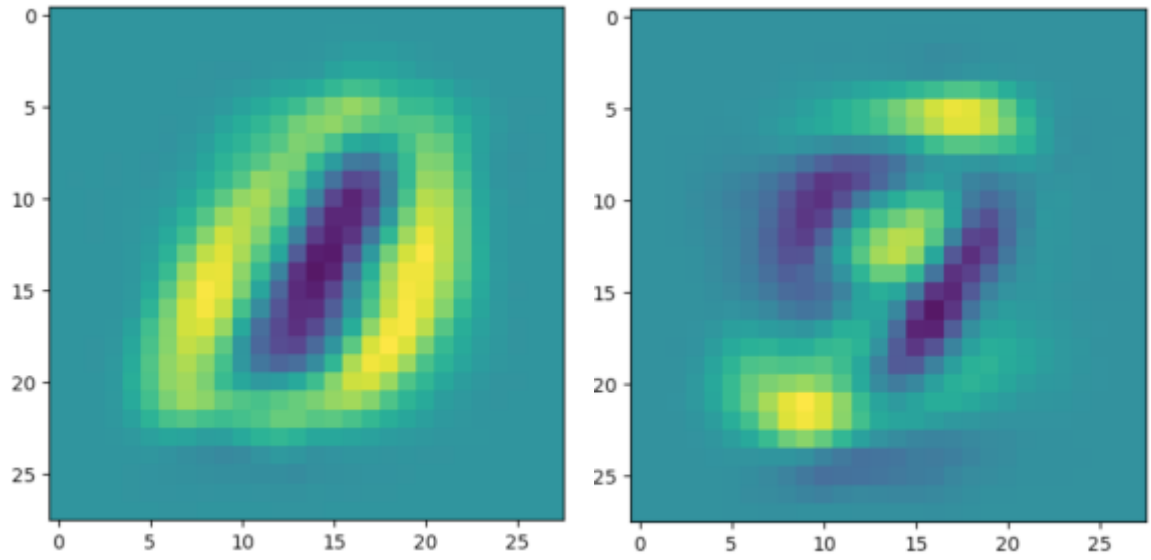
Goal: Find $Vx = \lambda x$

$$V^T Vx = V^T \lambda x = \lambda Vx$$

$$V\, V^T Vx = V\, \lambda Vx$$

If I manage to find the eigenvalues and eigenvectors corresponding to $V^T V$, when using V as a linear transformation to the eigenvector I got, it automatically gives us the eigenvector of V.

I conduct as the ways described above and get the eigenvectors with normalized to norm 1 and eigenvalues for V. It looks like this:

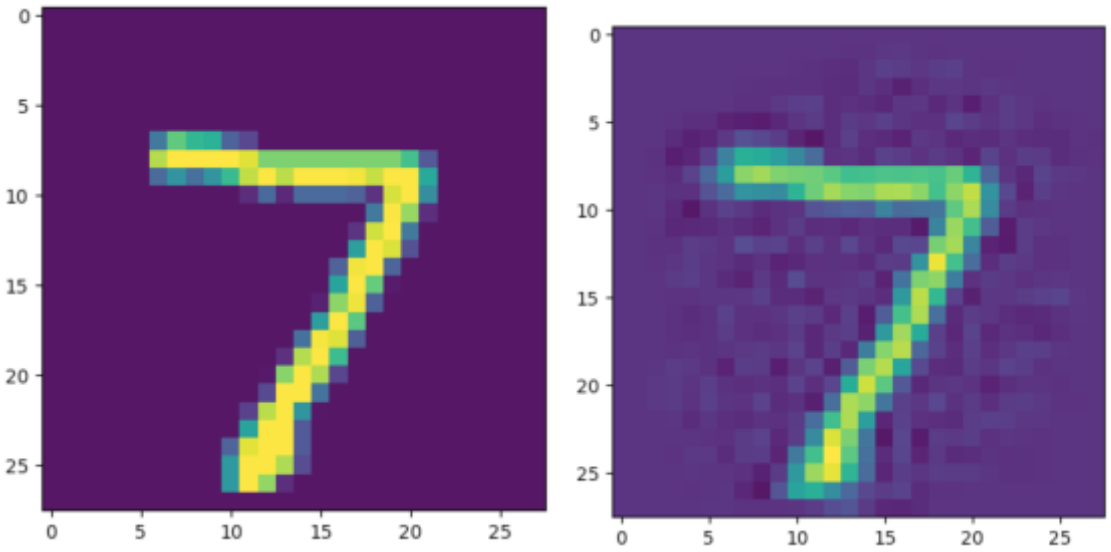


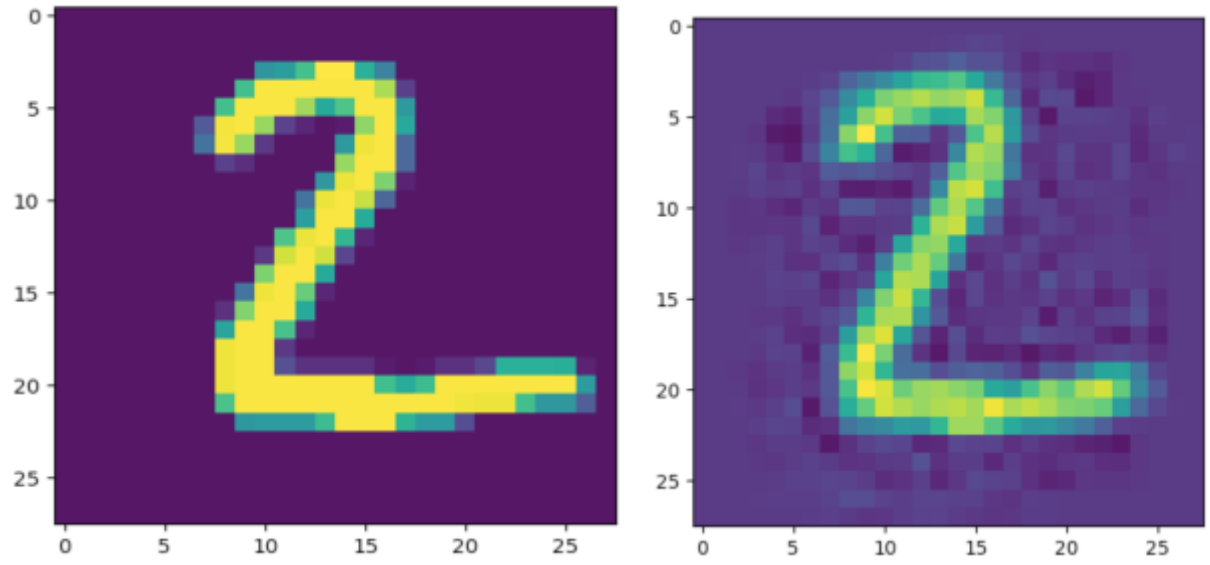
After this, I transfer all training image and test image use similar way, subtract the mean and times the eigenvector matrix so that both test data and train data are in a smaller dimension.

After transform to a smaller dimension, I implement the k-nearest-neighbor algorithm and sort the training image vector by the distance to the testing image vector and let them vote on those by the distance.

As the further the vector away, the less likely it will be that number. I let every image has different rights in voting depends on their ranking in distance by setting it to be 1/L, where L is the distance between the testing vector and the training vector. This algorithm takes the distance into consideration but also has a drawback that if most of the training data are of a single kind, the testing image will be mislabeled a lot. For example, if 91% of training data is representing 9 and 1% to represent 0 through 8 each and they spread uniformly across the space. No matter what training set data I met, it is still likely that I will classify it as 9. Maybe add coefficients to the count in the training set can help alleviate this problem.

From here, I can reconstruct the testing image. This is done by inverting the process before, let the vectors times the transpose of the transition matrix and then add the mean vector. Due to dimensions reduced before as we did not pick all eigenvectors but rather picking those associated with greatest eigenvalues, the reconstruction pictures are not of same quality as the original pictures. Below are the testing images before(left) and after(right) going through the whole processes. For more pair of pictures before and after the process, please refer to directories “testpicturebefore” and “testpictureafter”.



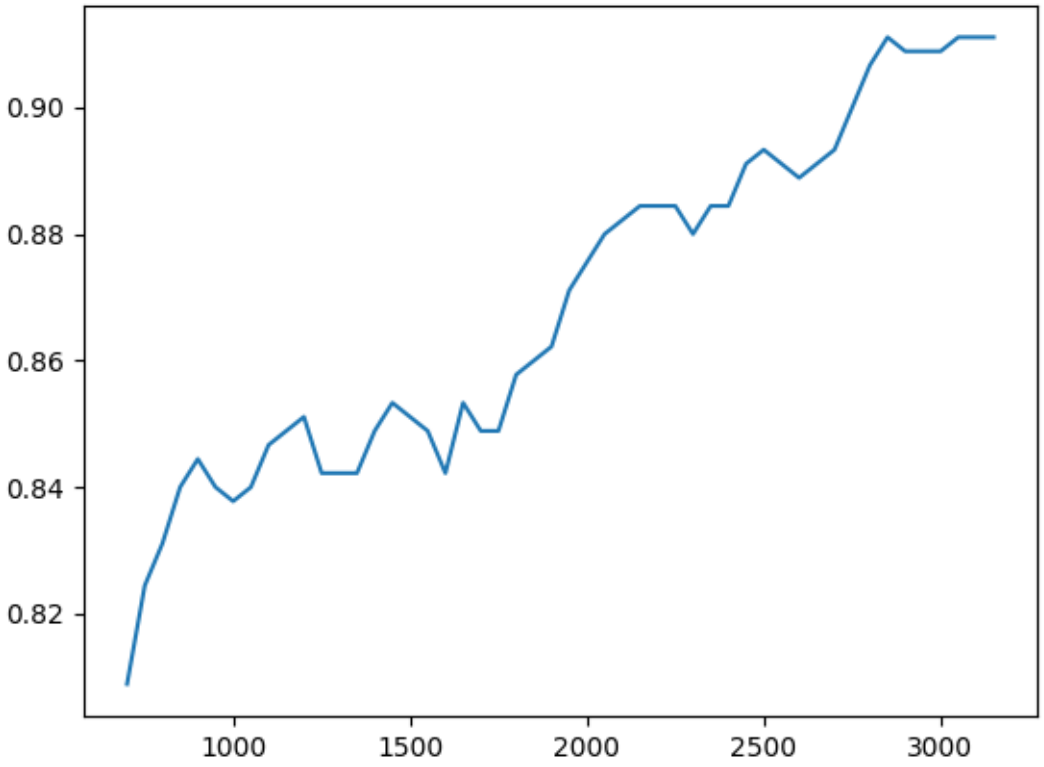


III Conclusion

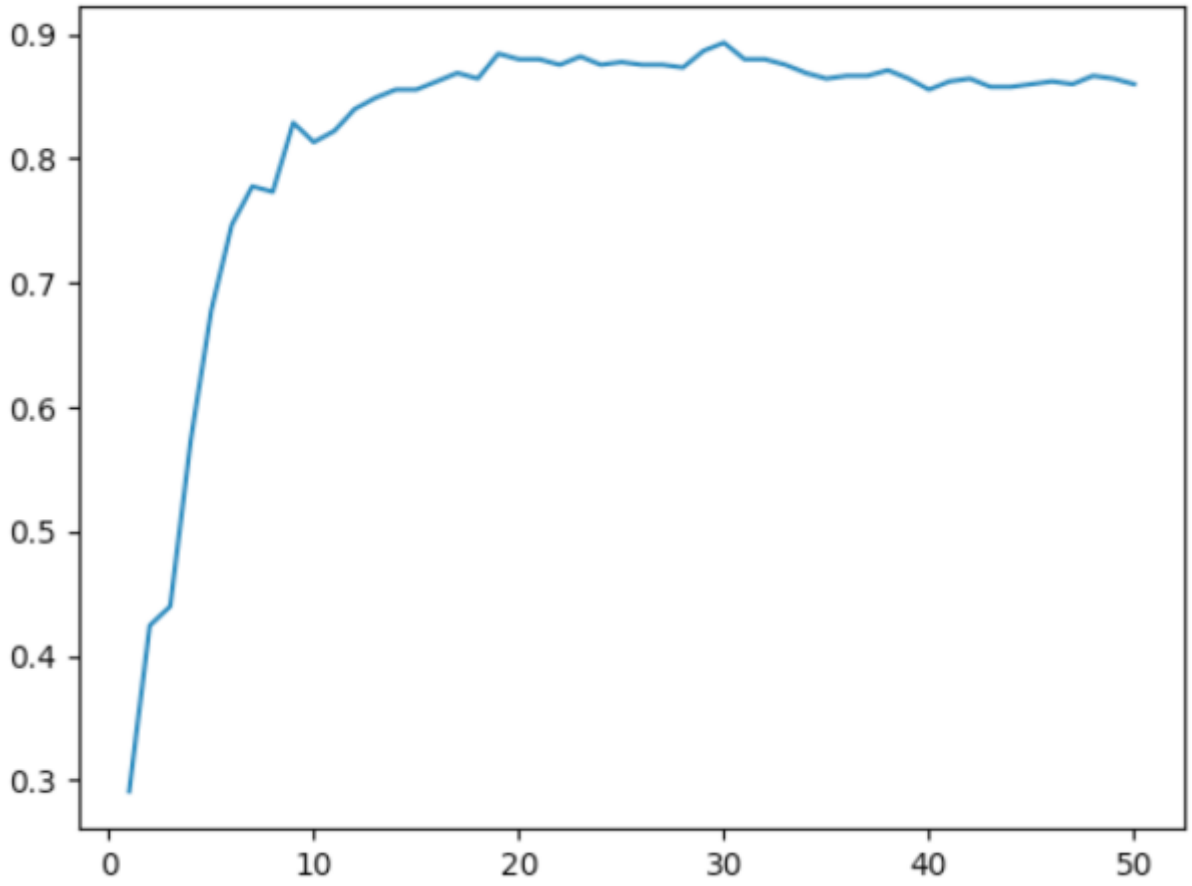
From the graph shown, I can see that our model generates pretty good predictions.

With more training data, the accuracy has an increasing trend, though drops at times. I take the training set of size increasing from 700 to 3200 with 50 as the gap of the step. For the testing set, it is 450 images.

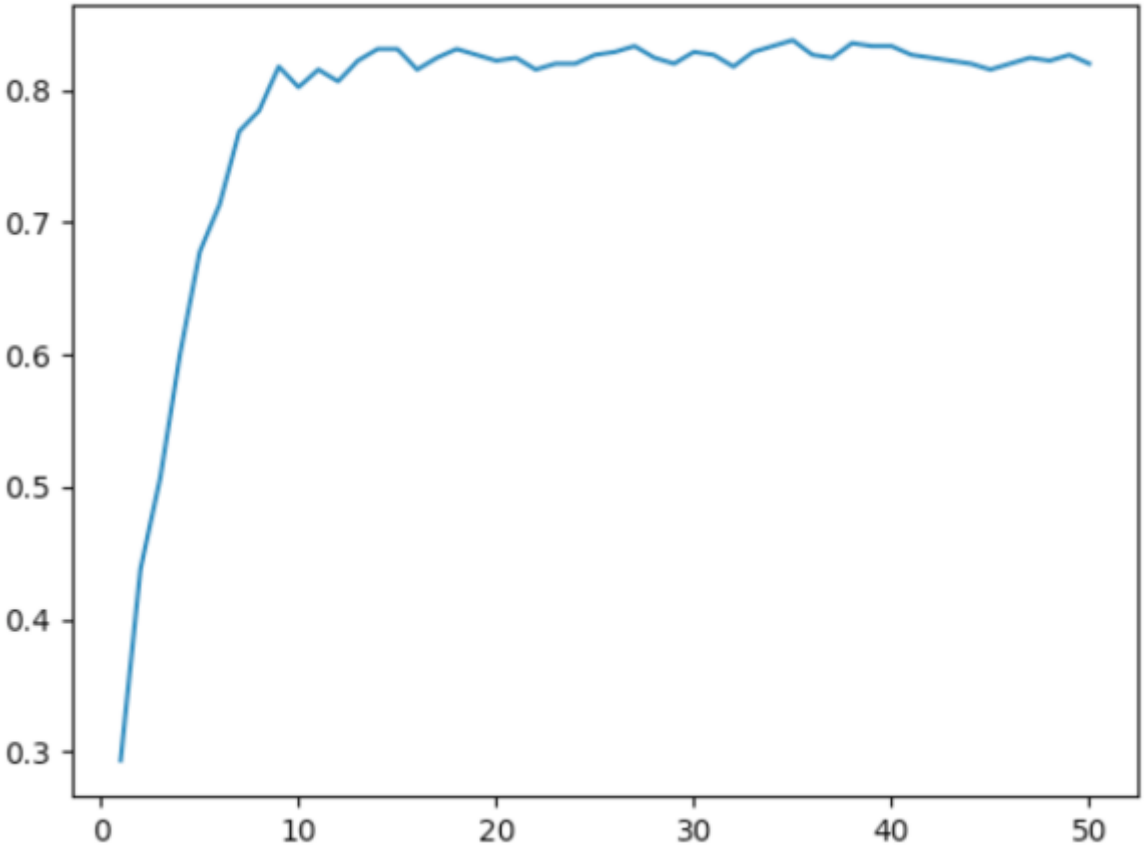
From the pictures listed below, I can find the accuracy was about 81% at the very start and rises up to about 91% at the very end. This is about a 10% increase in accuracy.



Also, for single predictions with the fixed number of training data and the fixed number of test data, I can change the number of eigenvectors selected. With more eigenvectors selected, the accuracy also gets better and better. For the graph below, it takes 1000 training images and 450 test images. The accuracy increases with the increasing of the eigenvectors selected.



For the knn, if I set every point of same weight and redo the experiment before, I get the following graph as accuracy with the increasing of number of eigenvectors.



As I can see, it is with similar trend but with lower accuracy as the graph before can reach about 0.9

IV Summary

In all in this lab, I implement PCA to analyze the major component and reduce dimensions. Later, I use a modified knn algorithms for classifying which eigendigit does the picture reflects. It shows that the increase in number of training pictures selected, the accuracy can increase sharply at the very start and keep stable later afterwards.

V Reference

Imshow functions:
https://stackoverflow.com/questions/25625952/matplotlib-what-is-the-function-of-cmap-in-imshow
Find Eigenvectors, eigenvalues transformations:
http://www.cs.utexas.edu/~dana/MLClass/NotesEigenfaces.pdf