Alex Navarro ajn853 Machine Learning HW1

**Introduction**

The task at hand is to create an image classifier capable of taking in an image representing a handwritten number, and reliably classify it as the correct numeral. The image sizes are fixed at 28x28 pixels of grayscale values, and a dataset of 10,000 images is available through the MNIST dataset available at  <http://yann.lecun.com/exdb/mnist/>.

**Method**

The chosen method for this project is a nearest neighbor comparison of eigenvector projection of candidate test images. The idea behind this method is that the training data is used to generate a set of eigenvectors for the space of handwritten digits. Each of these eigenvectors can be thought of like a unit vector in Euclidean space. Any point in the space should therefore be representable through a linear combination of these eigenvectors. Of course, there is no guarantee that a test image will fall into the vector space of the eigenvectors (in fact, it almost certainly will not), however a simple projection onto this vector space is enough to rectify this issue.  
   
Gathering Test Data

The data available through the MNIST dataset is not available directly in image form. The data is encoded in the IDX format. This format is not usable directly for computations in Python, and so the first step of the process is to decode the data. This was achieved in a fairly straightforward manner through the use of the Python standard library module “struct”. Through this, we were able to gather 4 pieces of data from the dataset:

* Labels of each image
* Height of each image in pixels
* Width of each image in pixels
* Pixel data

There was a total of 10,000 images in the dataset, so the runtime for this decoding was on the order of a few seconds.  
  
Generating the Eigenvectors

Once the data was read from the dataset, it needed to be converted into vector form to allow for the following vector operations to determine the eigenvectors. Since each image was 28x28 pixels, this meant that they could be represented with vectors of length 784. Each of these datapoints was appended on to the main data matrix as a row vector (I know it should be column vector, but I correct for it later). At this point, if we call each image row and the data matrix , then for data points we have

Now it is necessary to remove the offset from the data to simplify the calculations. Simply put, this means subtracting the mean from the data matrix, where the mean is calculated as the column-wise mean in the same shape as the input datapoint. We will call this offset matrix , where

Note that the transpose is simply to get the X matrix in the proper column-based format to match with the lecture notes, since I recorded the data in a row-based format.

Now given the matrix , the difference between the data and the mean, the covariance matrix of the data can be calculated as

It is the eigenvectors of this covariance matrix which form the vector space of the image classifier. Unfortunately, the covariance matrix is of size 784 x 784, which is computationally difficult, and would be infeasible for larger images of sizes. Luckily, there is an elegant solution. Using the results derived in the lecture, specifically that for a matrix , multiplied by the eigenvectors of are the same as the largest eigenvectors of . And the matrix is of size , which means that we can choose the size of the problem by limiting how many data points we import. We call these eigenvectors the eigenfaces of the images.

Image Classification

Once we have the set of eigenvectors from the matrix **,** we can create a projection matrix which projects data into the vector space of the eigenvectors. Specifically, we can project onto our choice of the largest eigenvectors, to determine the coordinates of the image in this space. This can be done for each image in the dataset, as well as any new image which is inputted after the eigenvector calculation is complete. If the matrix of eigenvectors is for the k largest eigenvectors, then the equation for a data image is

where is the projected coordinate point of the data in the eigenvector vector space. This means that we can compute these coordinates if can be inverted. For this, we can simply use the Moore-Penrose pseudo inverse , which can be easily implemented in Python as numpy.linalg.pinv.

Now images can be classified by selecting the test image with the smallest Euclidean distance from the projected coordinates of the unknown image. However, simply calculating the coordinates difference is not the entire story. Not all eigenvectors are created equal. The relative importance of each eigenvector in can be described by its eigenvalue, and as such it is important to calculate not only the coordinates of an image in the vector space, but its *weighted* coordinate. Hence, we multiply each term of the difference by its corresponding eigenvalue to get the weighted difference between two images.

**Results**

Test Data

The data was in images of 28x28 pixels. As they came from the MNIST dataset, they were inverted, so a simple inversion got them to the correct black-against-white form. The figure below shows the first 25 images in the dataset.

Text, shape

Description automatically generated

Eigenvectors

The following image shows the eigenvectors with the 16 largest eigenvalues. Note that this was taken from a training set of 400 images.

A picture containing text

Description automatically generated

If instead we were to use only 16 images for the training set, the 16 eigenvectors instead look like

A picture containing text, white

Description automatically generated

We can see that these eigenvectors look a lot more like actual numbers, since the number of building blocks available is significantly reduced. Note that there are no 3’s or 8’s in the first 16 images, and so they are not represented in the eigenvectors.

Reconstruction and Classification

If our projection into the eigenvector space is successful, then it should be possible to linear reconstruct a new image based on the eigenvector space coordinates calculated. The reconstructed image will not be a perfect match, since it is highly unlikely that it lay directly in the vector space, but there should be significant similarity with the original image. This works as a sort of “sanity check” that the calculations performed thus far are accurate. Note that it is also necessary to add back in the mean image in order to obtain a visually meaningful reconstruction. The following are 5 images that were successfully projected, reconstructed, and classified by the algorithm as well as 5 images that were misidentified. A training set of 400 images was used, as well as all 400 eigenvectors

Text, shape

Description automatically generatedText, shape

Description automatically generated

Now we can see that the algorithm works exactly as intended. The reconstructions are almost perfect (mostly because we used a huge number of eigenvectors) and the success rate is fairly high. Out of a total of 400 images test, 325 (81%) were successfully identified. Even the ones that were incorrectly identified still look a lot like the test image. But what happens if we reduce the number of training images? Well, let’s find out with a training set of 50 images:

Text, shape

Description automatically generatedA picture containing text

Description automatically generated

We can see that the reconstructions are much less crisp with less data to work with. This also explains why the accuracy of the image classifier is reduced in this case with only 242 (61%) of the test images identified correctly.

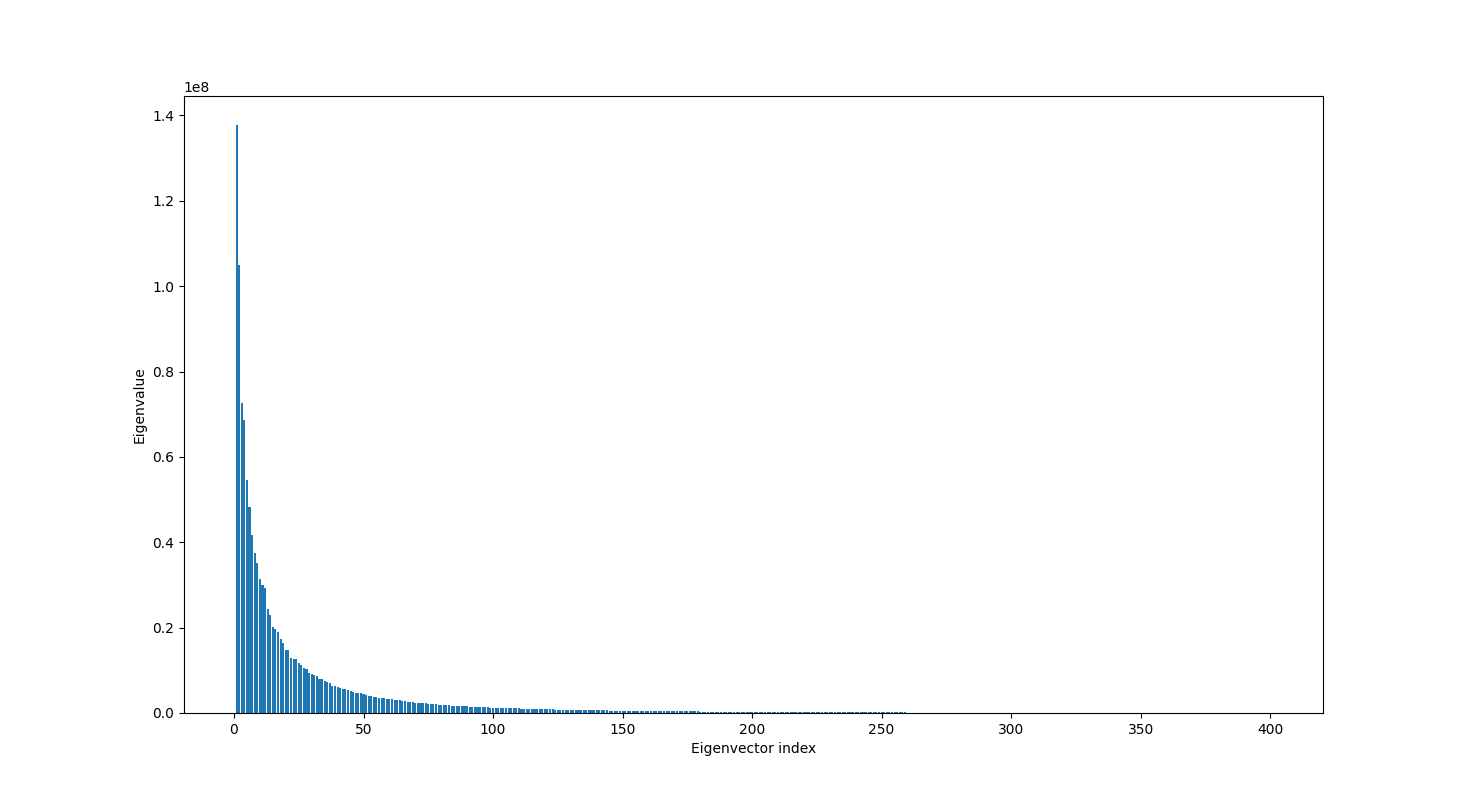
There is still one more case to test. What if we still used 400 images to train the data, but we only used the 50 most significant eigenvectors. Would the results be closer to the 400 image and 400 eigenvector case, or the 50 image and 50 eigenvector case? The results are below:

Text, shape

Description automatically generatedText

Description automatically generated

While the level of reconstruction is not quite as clear as the instance where all 400 eigenvectors were used, it is still a lot better than the base 50 eigenvector case. The success rate was 323 (81%), which only lost 2 images compared to the full case. The reason this works almost as well as the full 400 eigenvector case can be explained through the following plot



The 50th eigenvalue is only 3.25% the magnitude of the 1st eigenvalue. As such, the effect of any eigenvalues after the 50th are fairly insignificant, as we can see from the almost identical performance of the algorithm when they are omitted.

The effect of the number of training images on success rate is shown below. In this case, all eigenvectors are used.

Chart, line chart

Description automatically generated

The effect of image count seems to be in the general shape of a graph. The difference between using 50 images and 100 images was 34 more successful identifications, but when moving from 350 to 400 images, there were only 6 more successful identifications.

Alternatively, we can see the effect of including differing amounts of eigenvectors in the problem. In this case, the training set is fixed at 400 images, and the number of eigenvectors used is varied.

Chart

Description automatically generated

Immediately we can see that it does not take a lot of eigenvectors to get a good result. Even though we have 400 eigenvectors available to us, it only takes about 15 of them to reach the maximum effectiveness of the algorithm. Therefore, we can conclude that it is necessary to have a large number of training images, and not a large number of eigenvectors, to have an effective classifier. However, this does have a significant effect on runtime of both the training portion and the classification portion of the algorithm.