Classifying MNIST Digits Using Eigenvalues

September 2020

Chenkuan Liu

1 Introduction

The purpose of this project is to classify hand written numerical digits. The images we use come from the MNIST dataset. In general, we first reshape the training images into column vectors, and then use them to compute mean vectors and covariance matrix. Combined with principal component analysis (PCA), we can create images of "average digits", namely eigendigits. Then, during test stage, we project the test image onto the principal directions formed by eigendigits, and use k-nearest neighbor (KNN) to locate the nearby projections made by the training images. The labels that are closest to our testing projection are potential choices of our prediction.

2 Methods

2.1 Training

Each digit in MNIST dataset is represented by a 28×28 square matrix, with values ranging from 0 to 255. To ease our computation, we first only select a few images as our training set. We denote N as our sample size. After reshaping each square matrix to a 784×1 columns vector, we can use $\mathbf{m} = \frac{\mathbf{1}}{\mathbf{N}}\sum_{\mathbf{k=1}}^{\mathbf{N}} \mathbf{x^k}$ and $\Sigma = \frac{1}{N}\sum_{k=1}^{N}(\mathbf{x^k} - \mathbf{m})(\mathbf{x^k} - \mathbf{m})^{\mathbf{T}}$ to compute the sample mean vector and the sample covariance matrix.

There is an easy way to compute Σ. We can create a $784 \times N$ matrix A such that the ith column of A is just $\mathbf{x^i} - \mathbf{m}$. Note that we need to divide each entry of A by $\sqrt{N}$ and so AA^T is exactly the covariance matrix Σ. This step is helpful for us, because the size of A^TA is much less than the size of AA^T. Also, if $\mathbf{v}$ is an eigenvector of A^TA with eigenvalue μ ($A^TA\mathbf{v} = \mu\mathbf{v}$), then $A\mathbf{v}$ is an eigenvector of AA^T with the same eigenvalue ($AA^T(A\mathbf{v}) = \mu(A\mathbf{v})$). As a result, we can fairly quickly compute the N eigenvalues and eigenvectors associated with A^TA. In addition, these eigenvalues are also the N largest ones associated with the larger system AA^T. Using this technique, we can get the eigenvectors of AA^T associated with the N largest eigenvalues. They are the principal eigenvectors, and we can normalize them and rank them in order to form a new $784 \times N$ matrix V. Each column of V, after shaping back to a 28×28 square matrix, represents an eigendigit.

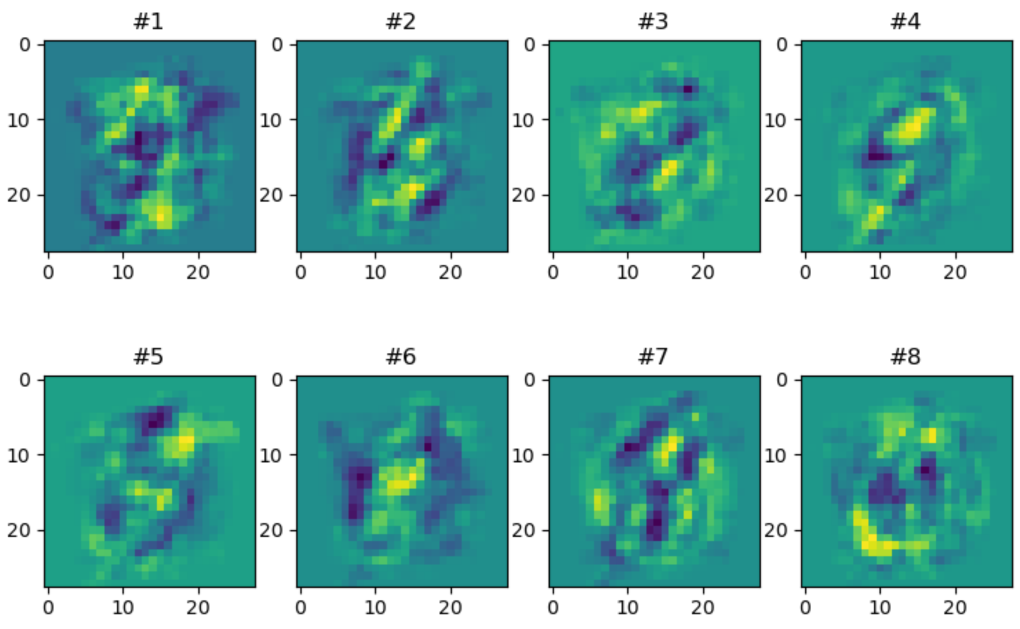


Figure 1: Images of Reshaped Eigenvectors with Descending Order of Eigenvalues

The Figure 1 above shows images of the first eight reshaped eigenvectors when setting sample size $N = 100$ (There are 100 of them in total). Each one is rather blurry and looks like many digits from 0 to 9 are overlapped altogether. They are our eigendigits and, in this case, represent the first eight principal directions of our training data.

2.2 Testing

To classify new images, we first need to form eigenvector spaces from our training images. For each training image, after reshaping and subtracting the mean, we compute its dot product with the columns of V. Specifically, from our previous computation, we have:

$$V = \begin{bmatrix} | & | & & | \\ A\mathbf{v_1} & A\mathbf{v_2} & \dots & A\mathbf{v_N} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \mathbf{u_1} & \mathbf{u_2} & \dots & \mathbf{u_N} \\ | & | & & | \end{bmatrix}.$$

Then, for each eigenvector space $\mathbf{\Omega_i}$ $(1 \leq i \leq N)$, we let

$$\mathbf{\Omega_i} = \begin{bmatrix} \mathbf{x_1} \cdot \mathbf{u_1} & \mathbf{x_2} \cdot \mathbf{u_2} & \dots & \mathbf{x_N} \cdot \mathbf{u_N} \end{bmatrix}^{\mathbf{T}}.$$

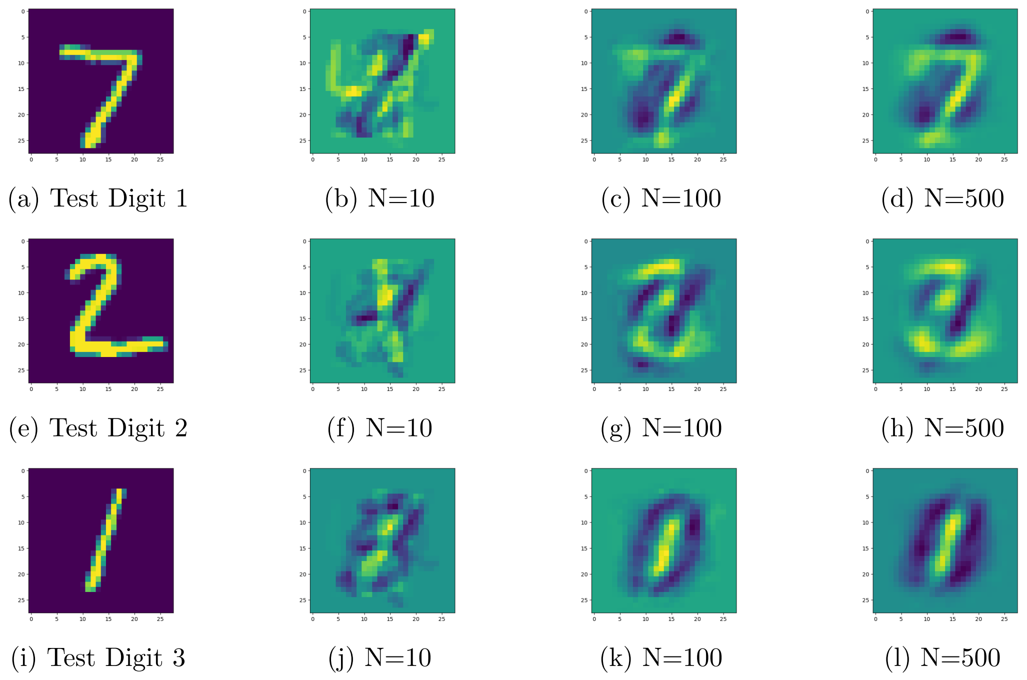
Now we have N eigenvector spaces. Each eigenvector space is in fact a column vector, with each entry represents the magnitude in one of the N principal directions. Then, given a new image $\mathbf{Y}$ (after reshaping and subtracting the sample mean), we compute $\mathbf{\Omega_Y} = [\mathbf{Y} \cdot \mathbf{u_1}\ \mathbf{Y} \cdot \mathbf{u_2}\ \dots\ \mathbf{Y} \cdot \mathbf{u_N}]^{\mathbf{T}}$. Next, from $(\mathbf{\Omega_1},\ \mathbf{\Omega_2},\ \dots, \mathbf{\Omega_N})$, we choose k closest ones to our $\mathbf{\Omega_Y}$ (we use L_2 metric here). We use KNN to find a group of eigenvector spaces from the k choices that are both close to $\mathbf{\Omega_Y}$ and corresponds to the same label. This label becomes our prediction for the testing image $\mathbf{Y}$.

3 Results

3.1 Fix Number of Eigenvectors

We first use all of our eigenvectors during testing. Some of the actual test labels and our reconstructed digits are displayed below.

After projecting onto the principal directions, we can actually use $V\mathbf{\Omega_y}$ to "compute back" and reconstruct our test images. This can help us get a visual understanding of the perfor-



(a) Test Digit 1 (b) N=10 (c) N=100 (d) N=500

(e) Test Digit 2 (f) N=10 (g) N=100 (h) N=500

(i) Test Digit 3 (j) N=10 (k) N=100 (l) N=500

Figure 2: (a)(e)(f): Actual Test Digits. (b)(f)(j): Reconstructed Digits with N=10. (c)(g)(k): Reconstructed Digits with N=100. (d)(h)(l): Reconstructed Digits with N=500.

mance of our model. In Figure 2, the first column on the left contains the first three digits in our test data, which are digits 7, 2, and 1. The second column contains the corresponding reconstructed images when training size $N = 10$. We see that they are rather unrecognizable. The predictions of the three images (b)(f)(j) give 1,1,1, with only the third one being correct. As the training size N increases, the reconstructed images become much clearer. The third column are reconstructed images under $N = 100$, we can see that the shapes of 7,2,1 become recognizable. In fact, images (c) and (k) gives correct predictions 7 and 1, but image (g) predicts 3 instead of the correct answer 2. On the last column, as N increases to 500, all of the three images (d)(h)(l) give correct predictions.

We fix the number of testing digits to 1000 and set $k = 4$ during KNN classification. By running the project multiple times with different training sizes, the model gives the following accuracy results, which are displayed in Figure 3 and Figure 4.

We see that the accuracy level increases significantly when we increase the training size from $N = 10$ to $N = 100$. After that, it increases with slower rates as N becomes larger. The

Training Size N	10	50	100	200	500	700	1000	2000
Accuracy	0.307	0.565	0.628	0.678	0.794	0.828	0.835	0.877

Figure 3: Accuracy Levels with 1000 Test Digits

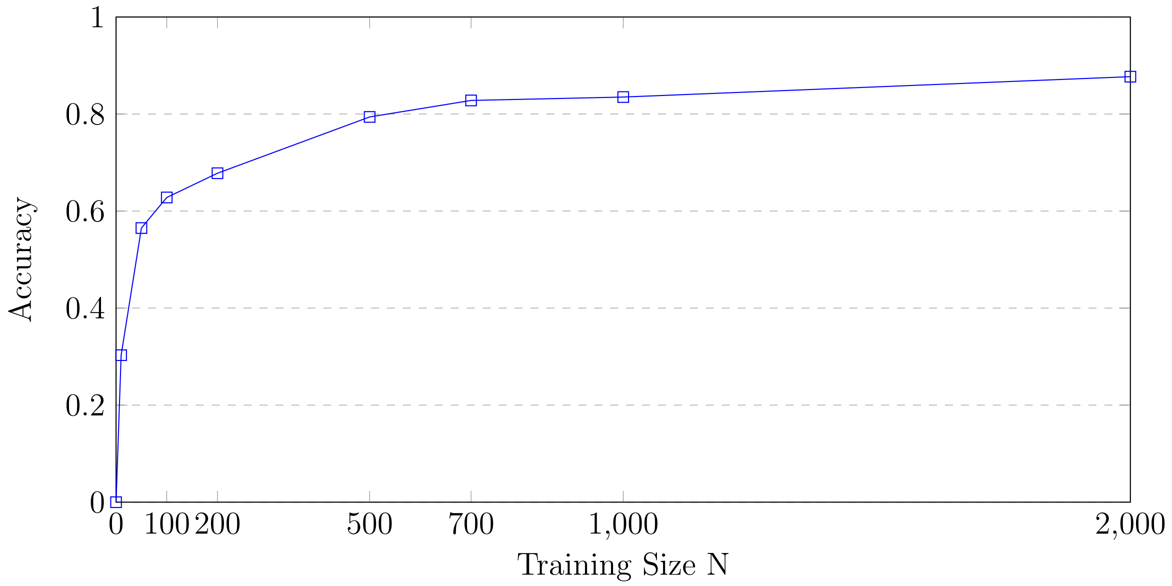


Figure 4: Plot of Accuracy versus Training Size

accuracy surpasses 80 percent when N reaches 700, and gets close to 88 percent when N becomes 2000. Note that, when N becomes larger than 784, we can compute AA^T directly since AA^T becomes the smaller matrix compared to A^TA, and we will always have 784 total eigenvectors when $N \geq 784$. However, when N becomes 700 or even larger, we should pay attention that the computation time also rises significantly.

3.2 Fix Training Size

Now, we fix our training size N to 700 and modify the number of eigenvectors we use when calculating L_2 distance during classification. We still use $k = 4$ in our KNN classification and 1000 test digits to evaluate accuracy. The results are listed below in Figure 5 and 6.

We see that, despite the slight decrease when we increase the number of used eigenvectors

# of Eigenvectors used	5	10	20	50	100	200	300	500	700
Accuracy	0.297	0.295	0.443	0.540	0.645	0.686	0.740	0.806	0.828

Figure 5: Accuracy Levels with Different Numbers of Eigenvectors

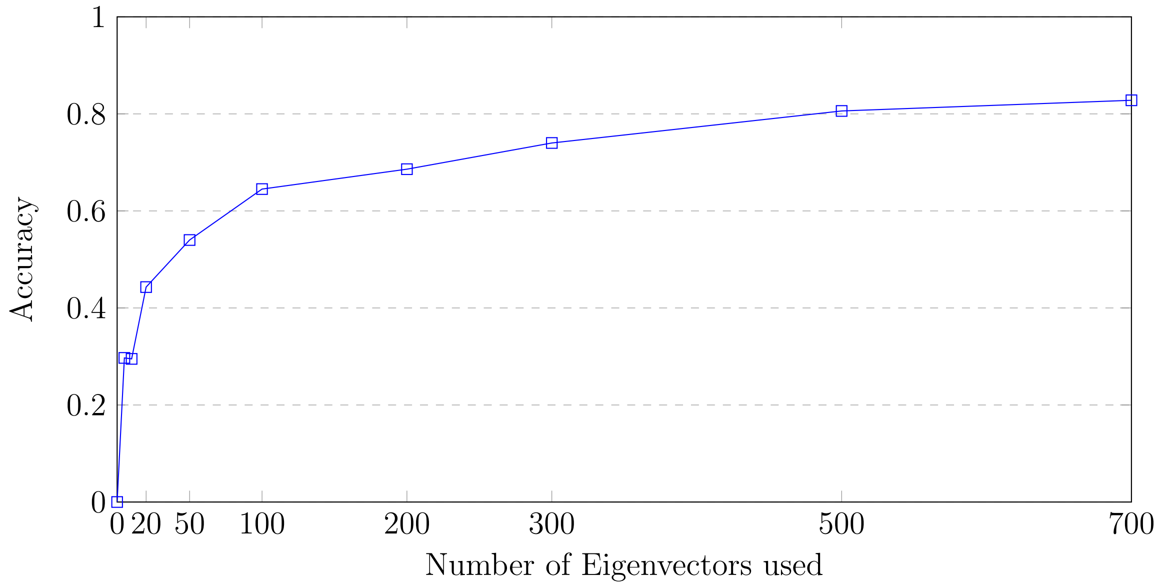


Figure 6: Plot of Accuracy versus # of Eigenvectors

from 5 to 10, the pattern of the plot is similar to what we got before in previous scenario in Figure 4, with significant increase of accuracy at small numbers of eigenvectors and steady improvement later on.

4 Discussion

In general, the results displayed above show that the degree of accuracy increases as we use more training data and include more eigenvectors during our computation. There are still many more things that can be explored within the project. So far, both the training and test images we use from MNIST dataset are starting from the first one. Namely, if we use $N = 500$ and 1000 test digits, we are actually using the first 500 training images and the first 1000 test images in MNIST. There could be possible variations if we choose the training set and test set randomly from the entire dataset. In addition, we are using L_2 metric, namely Euclidean distance, during our classification. It would also be interesting to investigate whether other metrics, such as L_1 norm and L_∞ norm, will affect our prediction accuracy.

Beside these possibilities, the parameter k in KNN, as well as KNN algorithm itself, are also topics worthy to pay attention to. In all of the experiments conducted above, k is fixed to 4. Though not displayed explicitly, when I increase k to 15 or even larger, the accuracy level actually decreases. It can be understood as "the abundance of choices hinders our predictions." However, such decrease of accuracy is also related to my specific implementation of KNN algorithm. Instead of calling packages, I write KNN using *numpy* and variations of bucket-sort. In my implementation, labels that appear more times always have higher priority than those that are closer but with fewer times. For instance, for $k = 15$, if we compute that the 15 closest labels to our test image projection are $[0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 1, 2, 9, 9, 9]$ (0 is the closest and 9 is the farthest), then my algorithm will pick 9 as prediction because it appears three times, instead of 0 which is closer but only appears two times. However, in reality, it seems that 0 might actually be the most likely real answer. This helps explain that small value of k may create better accuracy, because 9 will be removed from our consideration in the above instance if k is small. This also implies that how we design our KNN algorithm will have influence over the relation between values of k and prediction accuracy.