CS 391L Machine Learning Homework 1: Eigendigits

Yuege Xie, EID: yx4256, Email: yuege@ices.utexas.edu*

1 Introduction

Handwritten digits recognition is important and widely used in our daily life, such as recognizing zip codes for postal mail sorting and processing the amount of bank check. MNIST dataset developed by LeCun contains 50000 training and 10000 test images of hand written digits of numbers $0-9$. Each image is a 28×28 gray-scale $(0-255)$ image with label $0-9$. Figure 1 shows some examples of training and test images with their true labels. The project aims at training a classifier with training images to recognize unseen handwritten digits. We evaluate the classifier by its accuracy on test data. K-Nearest-Neighbors (KNN) is our target type of classifier, which uses majority votes of k nearest neighbors of the vector representation of an image to decide the category of the image. With different k, the value of votes varies.

The dimension of each digit image is $28 \times 28 = 784$, so it takes a lot of time to train and test a classifier using the original data, which is very computationally inefficient. Hence, dimension reduction methods like Principal Component Analysis (PCA) are crucial for improving the efficiency of classification. PCA is an unsupervised method for dimension reduction by extracting most important information using eigenvectors with top eigenvalues of the covariance matrix of training data matrix. After projecting training and test data on eigenspace generated from PCA, for example, dimension is 50 rather than 784, the speed is much faster than before.

Hence, this project uses PCA to find eigendigits, does dimension reduction according to eigendigits, uses projected training to train K-Nearest-Neighbors (KNN) classifiers and projected test images to show accuracy of each classifier.

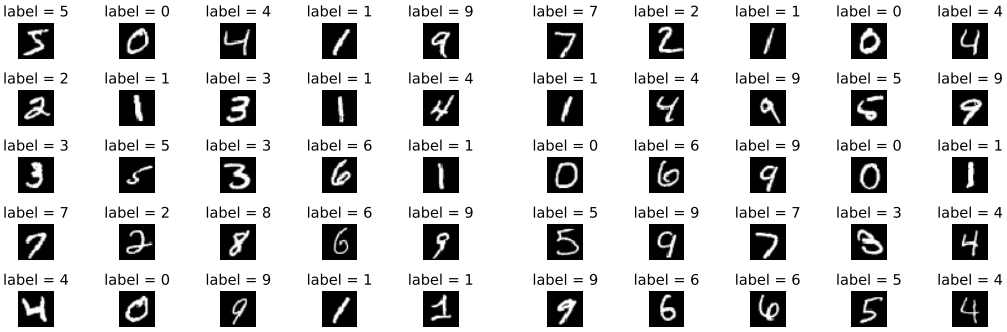

Figure 1: Left: Examples of training images; Right: Examples of test digits

2 Method

The method is explained in the introduction section and the detailed steps are as follows. The examples of top 25 eigendigits are shown in Figure 2. The examples of reconstruction of test images are shown in Figure 3. Since the sampled data are different at each time, we only show the construction of the same set of test images as in Figure 1 (Right) to compare the reconstruction results.

1. **Preprocess Data:** Load and preprocess (data is reshaped to a vector $x \in \mathbb{R}^{784}$ and divided by 255.0) data training and test data, and show some examples of training and test images;

*Oden Institute for Computational Engineering and Sciences, UT Austin.

2. **Sample training data:** randomly sample a subset of digits $X \in \mathbb{R}^{m \times n}$ from training data $X_{train} \in \mathbb{R}^{60000 \times n}$, for example, $m = 5000$.
3. **Feature extraction using PCA:**
   - **Get centralized data:** subtract mean vector μ from the data X using

$$\mu_j = \frac{1}{m} \sum_{i=0}^{m-1} X_{i,j}, \forall j = 1, 2, \ldots, n$$

$$A = X - \mu h$$

     where μh is the matrix that every row is μ to match the dimension of X.
   - **Calculate covariance matrix:** The covariance matrix is calculated by $cov = \frac{1}{m} A^T A$. Since the scalar $\frac{1}{m}$ does not affect the directions of eigenvectors, we can ignore it.
   - **Calculate eigenvectors and eigenvalues:** use the package "numpy.linalg.eigh"
   - **Sort:** sort eigenvalues and get the corresponding order of eigenvectors
   - **Choose top d eigenvectors as eigenspace V_d (eigendigits):** d is the dimension we keep. Since the eigenvectors calculated by the package are already normalized, we do not need to do normalization by ourselves. (See examples of eigendigits in Figure 2, where $d = 25$)

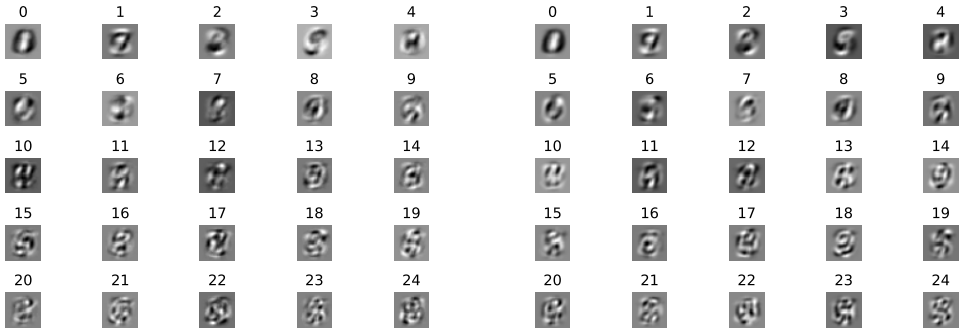

Figure 2: Top 25 eigendigits using random digits from training dataset. Left: $m = 1000$; Right: $m = 10000$.

4. **Project training and test data into eigenspace:**
   - **Subtract mean:** For any data matrix Y (training or test data), use $Y' = Y - \mu h$, where μh is the matrix constructed by mean vector of training data, the same as above.
   - **Projection:** project the data subtracted by mean on eigenspace using $Y_{proj} = Y' V_d$.
   - **Reconstruction:** reconstruct projected images from $X_{recon} = X_{proj} V_d^T + \mu h$. Note that this step is not necessary for training and test. But by reconstruction, you can see how much information is lost during the projection and investigate on the wrong predictions. (See Figure 3 for examples of reconstructions of test images)
5. **Train classifiers using k-Nearest Neighbor:** Use k-Nearest Neighbor method to train a classifier with projected training data and use majority vote to predict on projected test data.
6. **Compare and plot results:** compare accuracy of different classifiers trained by using different m, d and k.

3 Results

The code, plots and more example of eigendigits and corresponding reconstruction are in the appendix "hw1-eigendigits.ipynb". I show and analyze the plots in the following subsections.

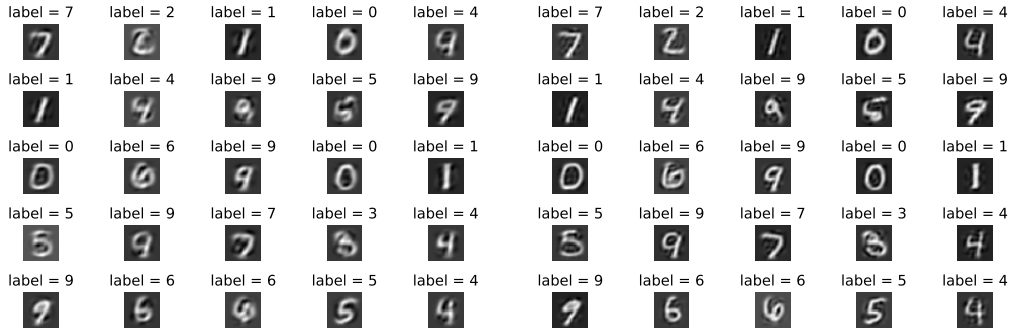

Figure 3: Reconstruction of first 25 test images using random $m = 10000$ digits from training dataset. Left: with top 20 eigendigits $d = 20$; Right: with top 50 eigendigts.

3.1 Accuracy of different numbers m of training data

As it shows in Figure 4, for fixed d and k (both $k = 3$ and $k = 5$), as the number of training data (m) increases, the accuracy increases. It means that using more data, the information contained in eigendigits is more accurate, and the classifier achieves higher accuracy. However, when m exceeds 10000, the speed of increase is slower than before. Hence, in the following plots, I mainly use $m = 10000$ for illustration.

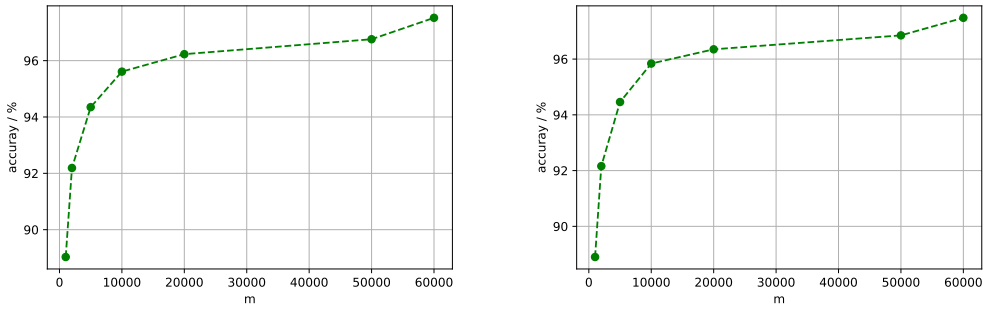

Figure 4: Test accuracy of different numbers m of training data with $d = 50$. Left: $k = 3$; Right: $k = 5$.

3.2 Accuracy of different dimensions d of eigenspace

As it shows in Figure 5, for fixed $m = 10000$ and k (both $k = 3$ and $k = 5$), as the number of top eigendigits (d) increases, the accuracy increases at first, achieves the highest accuracy around $dim = 50$, and then decreases a little bit slowly. At first, more eigendigits capture more information, so that the projected data can be separated well. However, when the dimension is too large, the points in higher dimension will be close. Since the KNN method depends on majority vote according to nearest neighbors, the classifier may be confused. Besides, the runtime of higher dimension eigenspace is larger than smaller one, which is called "the curse of dimensionality". Hence, for plots in other sections, I mainly use $d = 50$ for illustration.

3.3 Accuracy of different numbers k of nearest neighbors

Figure 6 shows the accuracy of different k with fixed $m = 5000$ and $d = 20, 50, 100$, respectively. When $d = 20$ or $d = 100$, $k = 5$ is a reasonable choice; when $d = 50$, $k = 1$ achieves the highest accuracy. Figure 7 shows the accuracy of different k with fixed $m = 10000$ and $d = 20, 50, 100$, respectively. In these cases, $k = 5$ or $k = 7$ is a reasonable choice. Hence, for plots in other sections, I mainly use $k = 3$ or $k = 5$ or $k = 7$ for illustration.

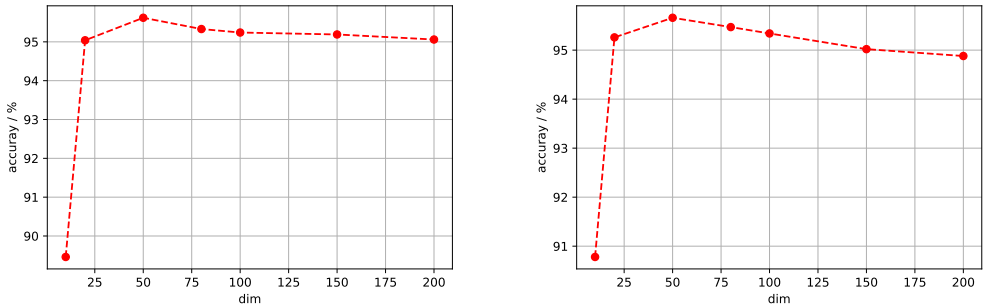

Figure 5: Test accuracy of different dimensions d of eigenspace with $m = 10000$. Left:$k = 3$; Right: $k = 5$

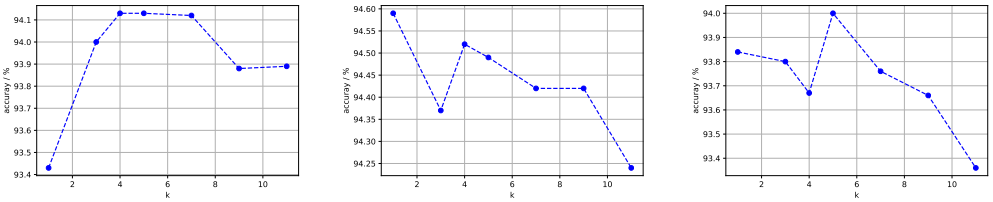

Figure 6: Test accuracy of different numbers k of nearest neighbors with $m = 5000$. Left:$k = 20$; Middle: $d = 50$; Right: $d = 100$.

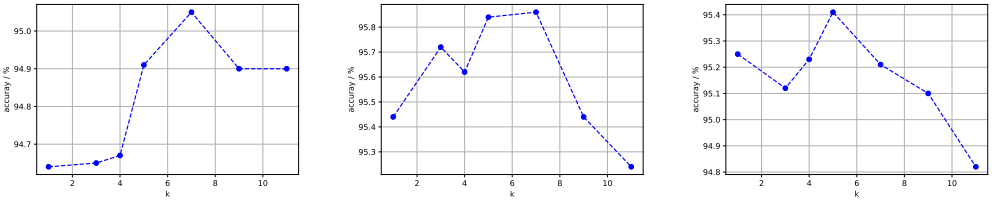

Figure 7: Test accuracy of different numbers k of nearest neighbors with $m = 10000$. Left:$k = 20$; Middle: $d = 50$; Right: $d = 100$.

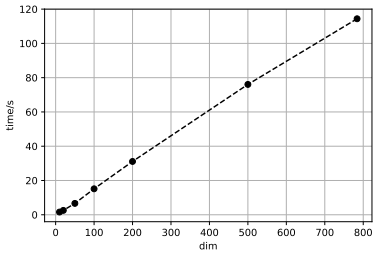

Figure 8: Run time of different dimensions d of eigenspace with $m = 10000$ and $k = 5$.

4 Summary

We use the accuracy (%) on test dataset to evaluate each KNN classifier. The analysis of plots are mainly in Section 3. In most cases, when m is larger than 5000 and dim is larger than 50, the test accuracy is usually larger than 90%. With proper choice, for example, when $m = 10000$, $d = 50$ and $k = 5$, the test accuracy is over 95%. In this case, we only use top 50 eigendigits to represent data, and the runtime is around $6.6s$ while for $d = 784$, the runtime is over $114s$. (Figure 8 shows that the runtime grows almost linearly as d increases.) Therefore, we verifies that the classifier trained by data projected with PCA dimension reduction is both accurate and computationally efficient.