CSE 546: MACHINE LEARNING HOMEWORK 3

DAVID P. FLEMING

Contents

Introduction

Please note that a copy of all the code I wrote to answer the questions in this assignment are included in my submission but also located online at `https://github.com/dflemin3/CSE_546/tree/master/HW3`. Some scripts require data, such as MNIST data, to run to completion and were not included on my github due to file size constraints. The MNIST data is included in the `Data` directory as python `.pkl` files as this compressed format gave me quicker load times for my scripts.

Overall, my code is structured as follows: There are three main directories included in my submission: `DML`, `Data` and `HW3`. `HW3` contains all the scripts used to run my analysis. For example to reproduce the answer for Question 1.2, one would run `python hw3_`1.2.*py*. All scripts have relative file paths so the code should run and have detailed comments to describe functionality. In addition, the scripts have flags near the top to control functionality. By default, I set all flags to true so the script performs the entire analysis and plotting. The `Data` directory

Date: November 21st, 2016.

contains both the MNIST dataset. The grader should be able to run my homework scripts without altering this directory.

The `DML` directory contains all the auxiliary files used to do the computations in the homework scripts and has a logical hierarchy. For example, the directory `optimization` contains the file `gradient_descent.py` while contains both my batch gradient descent and stochastic gradient descent implementations. The directory `data_processing` contains the script `mnist_utils.py` which contains my functions used to load and work with the MNIST data. The directory `classification` contains the file `classifier_utils.py` which contains all things related to binary and softmax classification including the gradients for each respective method for use with a gradient descent algorithm. The `validation` subdirectory contains `validation.py`. This file contains all my loss functions such as 0/1 loss and also my implementations for regularization paths for both linear regression and logistic and softmax classification using gradient descent. Finally, the `regression` directory contains all utilities for a normal or multi-class regression. In particular, this directory contains the file where my ridge regression implementation lives, `ridge_utils.py`.

In each section, I try to be explicit with what files I used to perform the computation including the path from the `DML` directory for ease of grading.

Question 0: Collaborators

I collaborated with Matt Wilde, Serena Liu, and Janet Matsen for various questions on this assignment.

Question 1: PCA and reconstruction

In this question, I derive matrix algebra relations and I use my PCA implementation to reduce the dimensionality of the MNIST dataset and visualize the results. The code used to solve this question is in the following attached files: `hw3_1.2.py, hw3_1.3.py, hw3_1.4.py` in the `HW3` directory and `classification/classifier_utils.py, regression/regression_utils.py, validation/v` `data_processing/mnist_utils.py` in the `DML` directory.

1.0: Details of my PCA implementation. My PCA implementation performs a SVD on the MNIST training datat X to derive the principal components and the eigenvectors of the empirical covariance matrix, Eqn. 3. Performing PCA using a SVD gives me the following matrices: u, s, and v where the columns of v are the eigenvectors of Eqn. 3. The eigenvalues of Eqn. 3 are given by the square of the singular values divided by the length of the dataset, $\lambda = s^2/N$.

To transform the dataset to a lower dimensional space using the first l principal components, the first l columns of v, I evaluate

$$X_{trans} = Xv \tag{1}$$

since X is $N \times d$ and the truncated v is $d \times l$ resulting in a lower-dimensional $N \times l$ X_{trans}. I reproject my lower-dimensional transformed data back into"physical" space via

$$X_{approx} = X_{trans} v^T \tag{2}$$

since X_{trans} is $N \times l$ and v^T is $l \times d$ resulting in a $N \times d$ matrix. Note that $X_{approx} \neq X$ for $l < d$ principal components since using less than the total feature number of principal components instead yields an approximation of the original data when projected back into the original space.

Note that for all questions, I first centered the training data before running PCA. Specifically, for each feature (column) of X, I subtracted off its mean. In addition to the textbook suggesting this as a good practice, I found that it yielded better performance and hence do so for all PCA questions in this homework set in less explicitly stated otherwise.

1.1: Matrix Algebra Review.

$$\Sigma = \frac{1}{N} \sum_i x_i x_i^T \tag{3}$$

TODO

1.2: PCA. Here I performed PCA on the MNIST training data and retained the first 50 principal components and singular values. The code used to answer this question is located in files: `hw3_1.2.py` in the `HW3` directory and `pca/pca.py` in the `DML` directory.

1.2.1. The eigenvalues $\lambda_1, \lambda_2, \lambda_{10}, \lambda_{30}, \lambda_{50}$ are 332719.1, 243279.9, 80808.5, 23685.7, and 11035.3, respectively. The sum of all eigenvalues is 3428445.4. Again, I calculate $\lambda = s^2/N$ for singular value s and dataset size N.

1.2.2. I plot the fractional reconstruction error for $k \in [1, 50]$ for the k principal components in Fig. 1. The fractional reconstruction error is defined as

$$F = 1 - \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{d} \lambda_i} \tag{4}$$

for d total features and k principal components. Note how the first principal component is the largest eigenvalue as shown above.

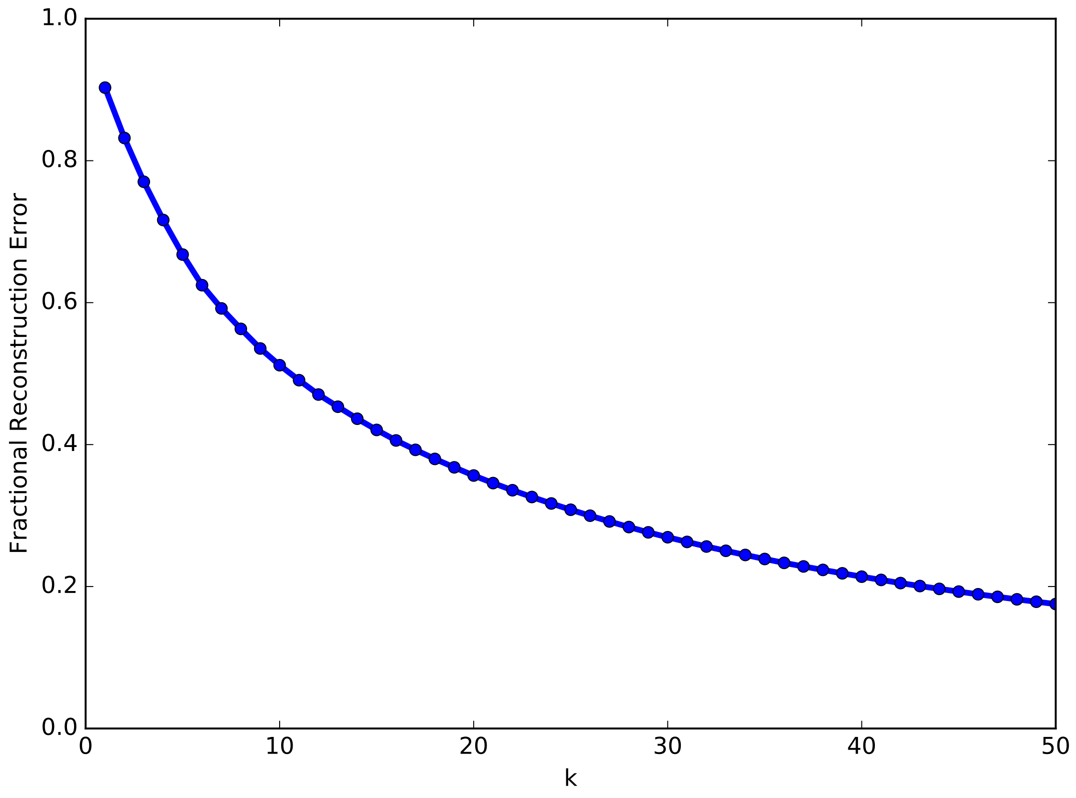

FIGURE 1. Fractional reconstruction error as a function of k principal components out of d total dimensions for $k \in [1, 50]$.

1.2.3. The first eigenvalue, and its corresponding principal component, represent the component of the data that captures the most variance. That is, the first eigenvalue represents the mean covariance of the data.

1.2.4. I plot the fractional reconstruction error for $k \in [2, 50]$ for the k principal components in Fig. 2. As expected, we see that the absence of the first eigenvalue demonstrates its importance.

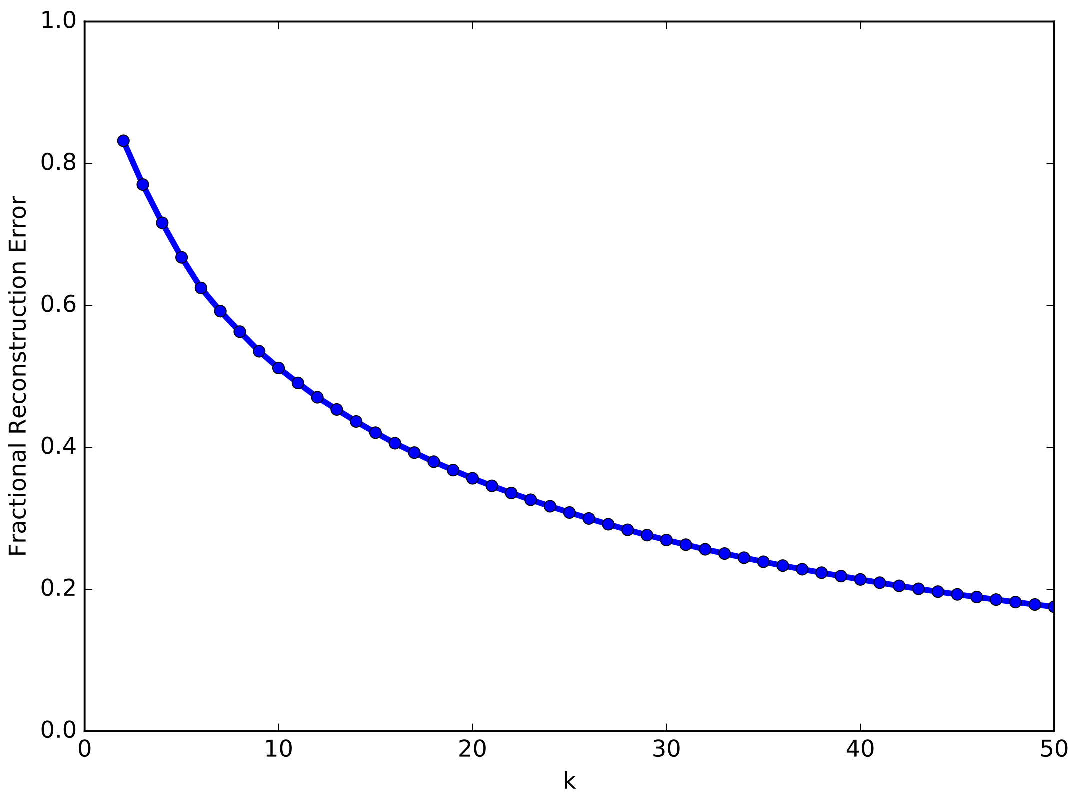

FIGURE 2. Fractional reconstruction error as a function of k principal components out of d total dimensions for $k \in [2, 50]$.

1.3: Visualization of the Eigen-Directions.

1.3.1. I plot the first 16 eigenvectors in Fig. 3 as images. To do so, I reshaped each vector into a 28×28 pixel image. For each image, I also annotate it with its number such that the second eigendirection would have a "2" in its image. I plotted 16 instead of 10 since 16 gives a nice symmetric collection of subplots.

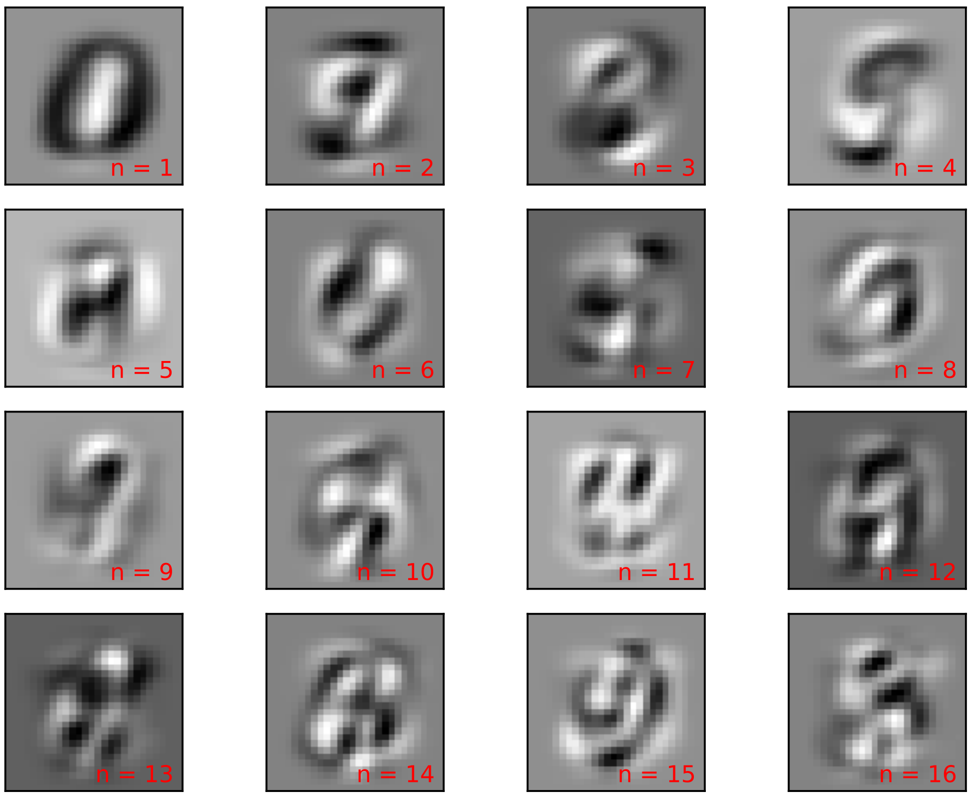

Figure 3. Visualization of the first 16 eigendirections as 28×28 pixel images. The respective eigendirection number is given in red in a subplot's lower righthand corner.

1.3.2. Each individual eigendirection, or principal component, captures a fundamental basis vector in an orthogonal set which each explain a decreasing variation in the data with increasing principal component number. That is, the first principal component accounts for the most variability in the data while subsequent components account for less and less variability. This intuitive picture makes sense upon examining Fig. 3. The first four components themselves appear to represent the numbers 0, 9, 3, and 5, respectively. Although the components do not explicitly represent those numbers, instead the represent characteristics of numbers in the dataset that are commonly shared. Many of the digits share features with 9, 3, and 5, for example, like their openings on the lefthand side of the digit and the solid vertical line on the righthand side of the digit.

1.4: Visualization and Reconstruction.

1.4.1. Here, I choose 6 unique digits from the MNIST training set and plot them in Fig. 4. I annotate each digit's image with its label in the lower-righthand corner. I chose 6 digits instead of 5 for symmetry's sake.

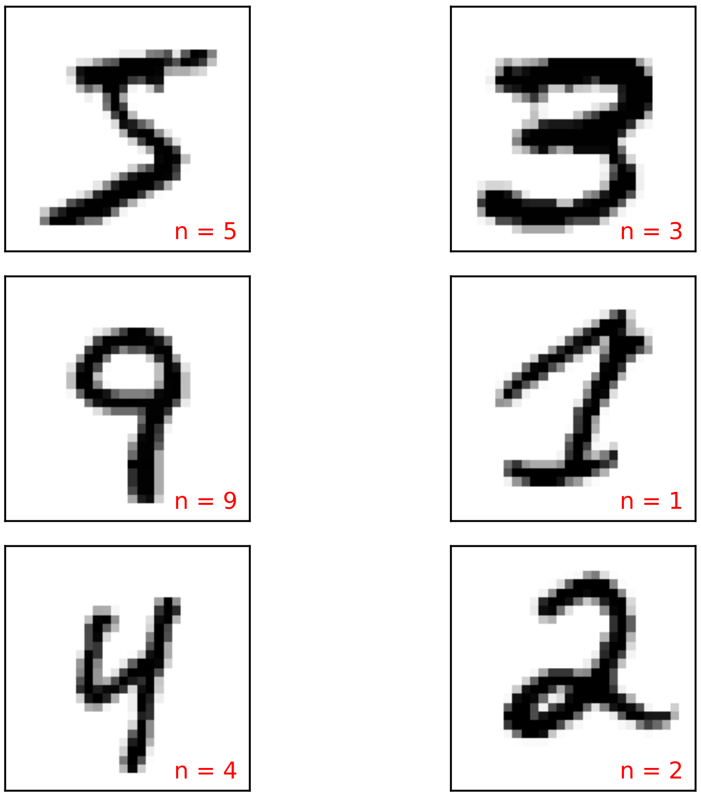

FIGURE 4. Visualization of 6 random, unique digits from the MNIST training set. The image label is displayed in red in the lower-righthand corner of each image.

1.4.2. Now I perform PCA on the MNIST training set keeping the first 100 principal components. Using this fit and the same digits displayed in Fig. 4, I reconstruct the digits using 2, 5, 10, 20, 50, and 100 principal components using the reconstruction $X_{recon} = Xvv^T$. The reconstructions are displayed in Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, and Fig. 10. For each reconstructed image using k principal components, k is shown in red in the lower-righthand corner.

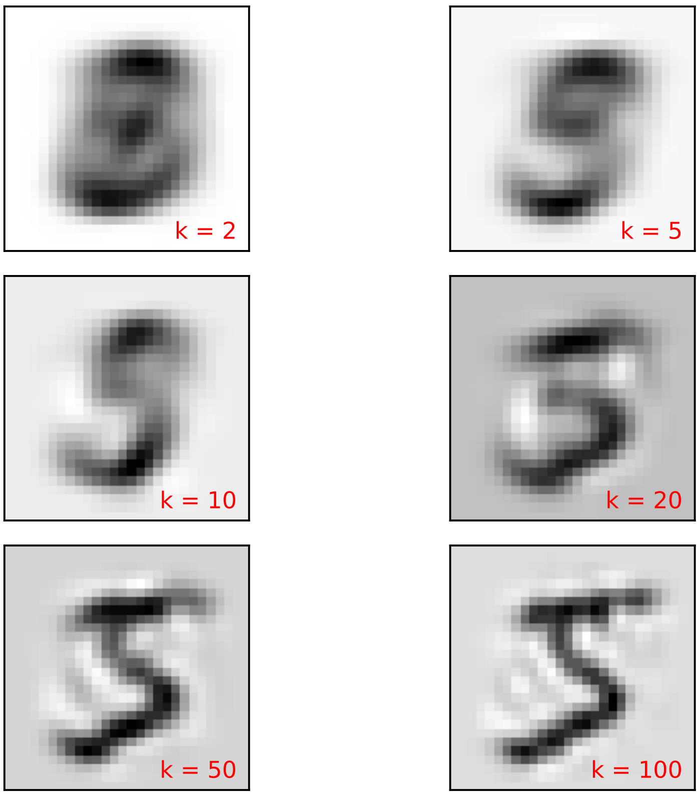

FIGURE 5. Reconstruction of a number 5 from the MNIST data set (see Fig. 4 using $k \in [2, 5, 10, 20, 50, 100]$ principal components where the given k is denoted in red in the lower-righthand corner of the respective subplot.

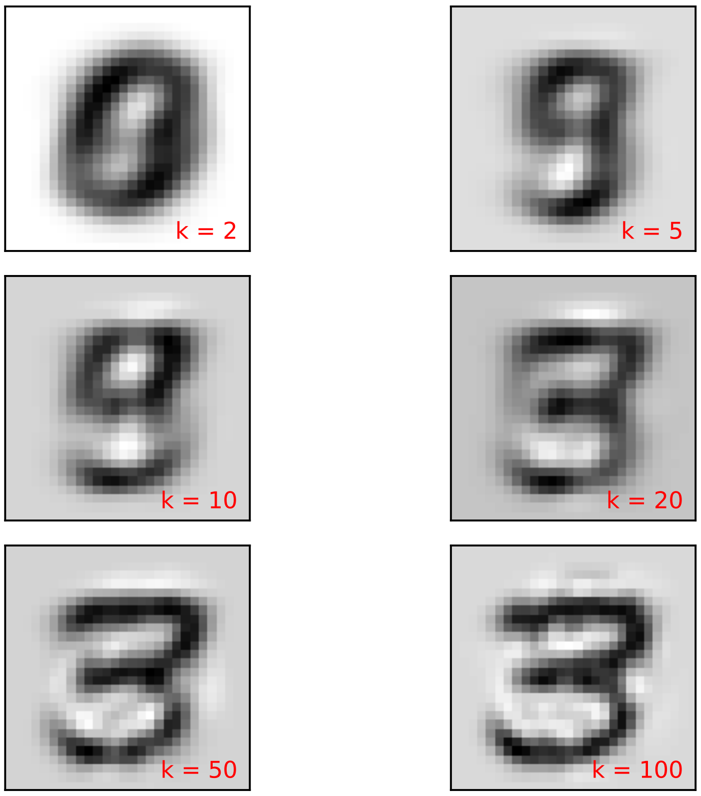

Figure 6. Reconstruction of a number 3 from the MNIST data set (see Fig. 4) using $k \in [2, 5, 10, 20, 50, 100]$ principal components where the given k is denoted in red in the lower-righthand corner of the respective subplot.

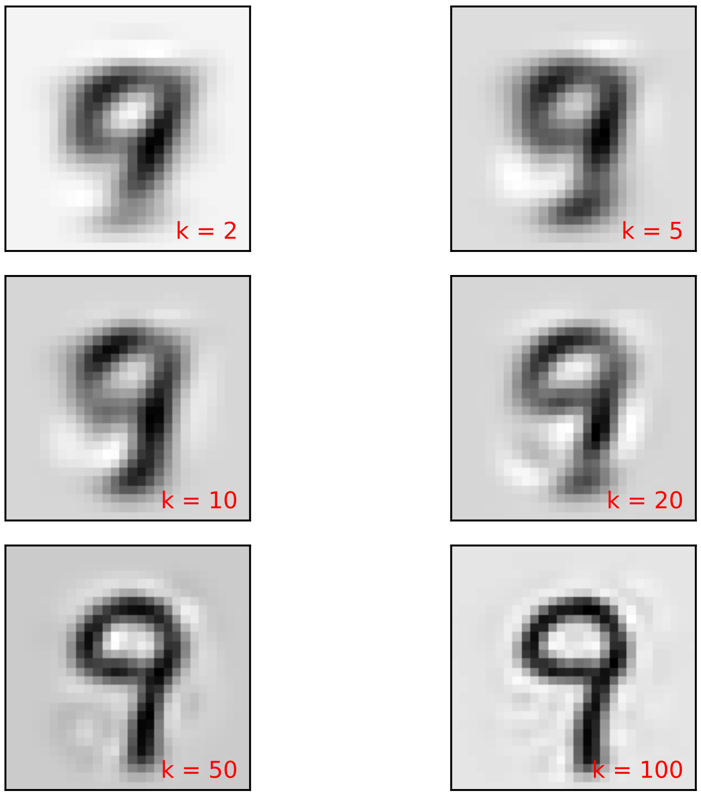

Figure 7. Reconstruction of a number 9 from the MNIST data set (see Fig. 4) using $k \in [2, 5, 10, 20, 50, 100]$ principal components where the given k is denoted in red in the lower-righthand corner of the respective subplot.

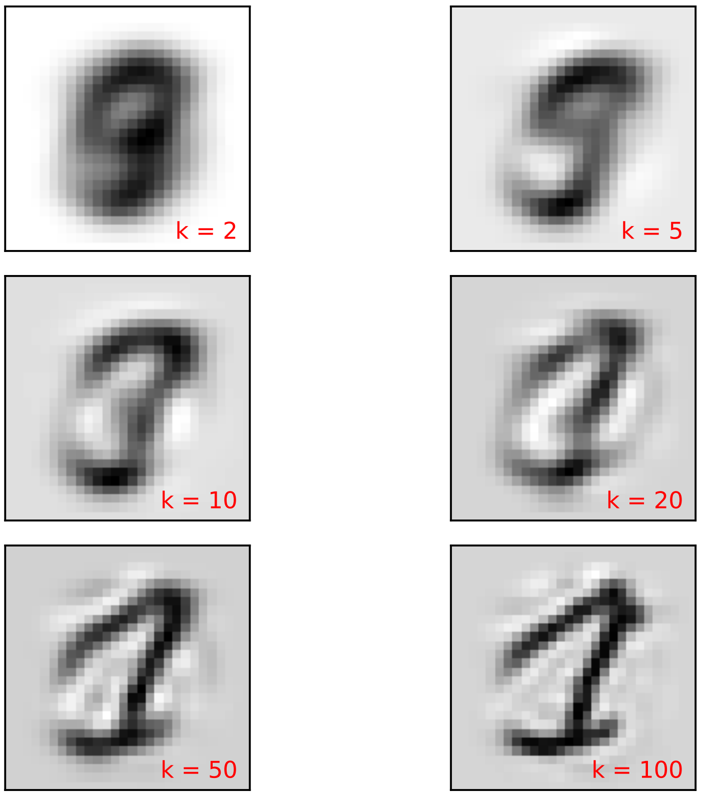

FIGURE 8. Reconstruction of a number 1 from the MNIST data set (see Fig. 4) using $k \in [2, 5, 10, 20, 50, 100]$ principal components where the given k is denoted in red in the lower-righthand corner of the respective subplot.

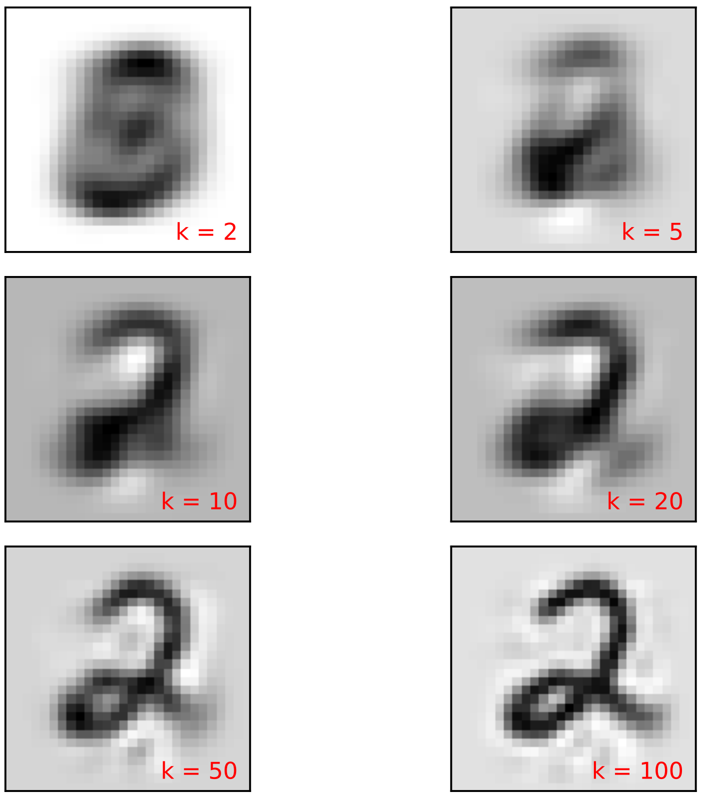

FIGURE 9. Reconstruction of a number 2 from the MNIST data set (see Fig. 4) using $k \in [2, 5, 10, 20, 50, 100]$ principal components where the given k is denoted in red in the lower-righthand corner of the respective subplot.

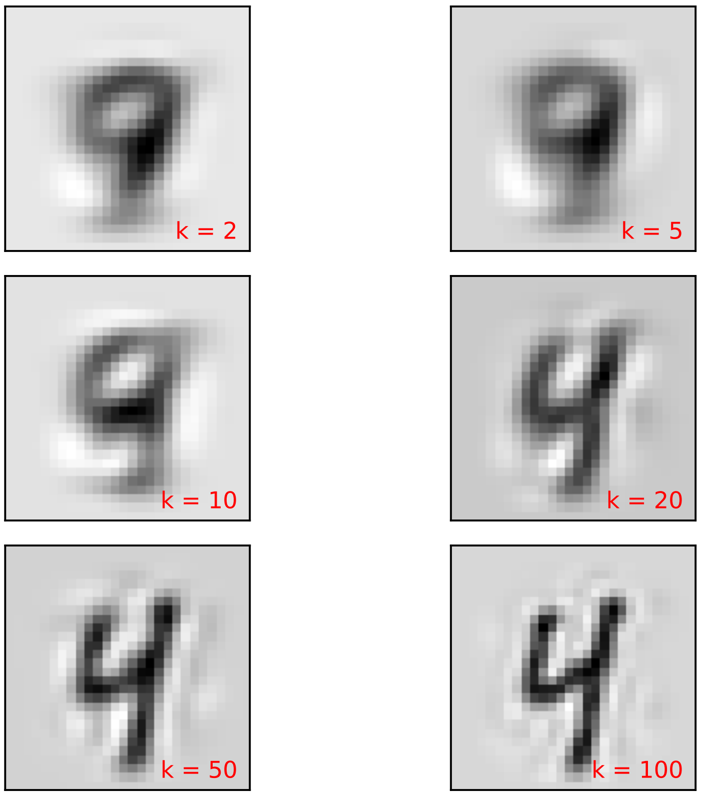

FIGURE 10. Reconstruction of a number 4 from the MNIST data set (see Fig. 4) using $k \in [2, 5, 10, 20, 50, 100]$ principal components where the given k is denoted in red in the lower-righthand corner of the respective subplot.

1.4.3. Broadly, the images reconstructed using at least $k = 20$ principal components appear to be a good, albeit grainy, approximation of the true image. Some images, in particular the reconstructed 9 and 5, are decently approximated using 10 principal components. Other digits that have large scatter in how they are drawn, such as 4s, require more principal components to adequately approximate the true image. All digits reconstructed with at least 50 principal components are good approximations of the original images. As expected, the more principal components used, the better the approximation becomes since as $k \to d$, we expect the reconstruction to become equal to the original X.

5: K-Means

In this question, I implement the K-means clustering algorithm and apply it to the MNIST dataset. The code used to solve this question is in the following attached files: `hw3_5.1.py, hw3_5.2.py` in the `HW3` directory and `clustering/kmeans.py, validation/validation.py, data_processing/mnist_utils.py` in the `DML` directory.

5.0: K-Means Algorithm Implementation. My K-means algorithm follows the algorithm presented in the Murphy textbook. First, I initialize my k clusters using k random samples from the MNIST training set to make a $k \times d$ cluster matrix μ for d features. Each iteration, I perform the following E and M steps. In the E step, I assign a label z to each sample X_i subject to

$$z_i = \text{argmin}_k ||X_i - \mu_k||_2^2 \tag{5}$$

which assigns a point X_i to the closest cluster center μ_k according to the squared Euclidean distance. After passing through the training set and assigning each point to its nearest cluster center, I perform the M step in which I calculate the new cluster centers to be the centroids of the samples in each cluster as follows

$$\mu_k = \frac{1}{N_k}\sum_{i=1}^{N_k} X_i \tag{6}$$

where there are N_k samples in the kth cluster. I call the algorithm converged once all the samples' labels no longer change.

5.1: Run the algorithm. I ran the K-means algorithm with $k = 16$ on the full MNIST training set. As stated above, I initialized my centers mu_k using k random samples from the training set.

5.1.1a. In Fig. 11, I plot the squared reconstruction error versus iteration number for $k = 16$ clusters. I compute the squared reconstruction error as

$$\text{SRE} = \frac{1}{N}\sum_{j=1}^{k}\sum_{X_i \in \mu_j} ||X_i - \mu_j||_2^2 \tag{7}$$

where μ_j is the cluster X_i belongs to.

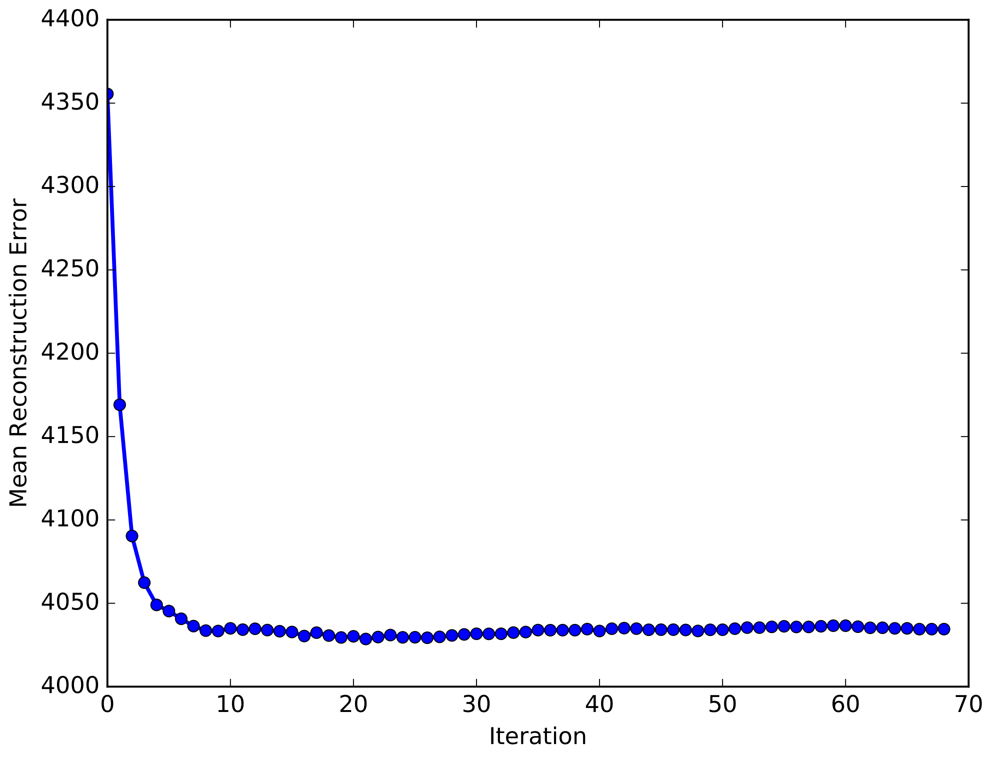

FIGURE 11. Squared reconstruction error versus iteration number for $k = 16$ clusters.

The mean squared reconstruction error reaches near its local minimum after about 10 iterations and does not change significantly over the next 60 iterations it takes the algorithm to converge. The final 60 or so iterations likely involved just a few points flipping labels, explaining why the error did not change appreciably.

5.1.1b. Let us define the number of assignments for a mean to be the number of samples assigned to that mean's cluster. In Fig. 12, I plot the number of assignments per cluster for $k = 16$ clusters ordered in descending order.

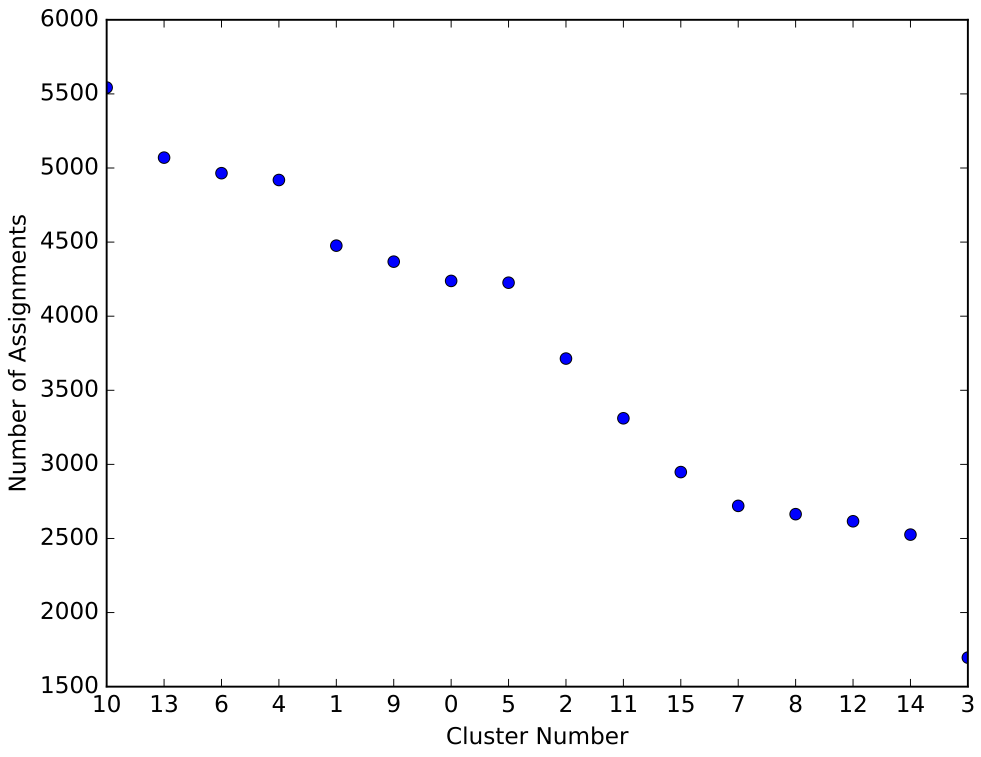

FIGURE 12. Number of assignments per cluster for $k = 16$ clusters ordered in descending order.

As can be seen in the above figure, cluster 10 contains the most members at about 5,500 while cluster 3 contains the least with around 1,750 members. I note that cluster number does not in general equal the MNIST digit label.

5.1.1c. In Fig. 13, I visualize the $k = 16$ centers my K-means algorithm learned on the MNIST training set.

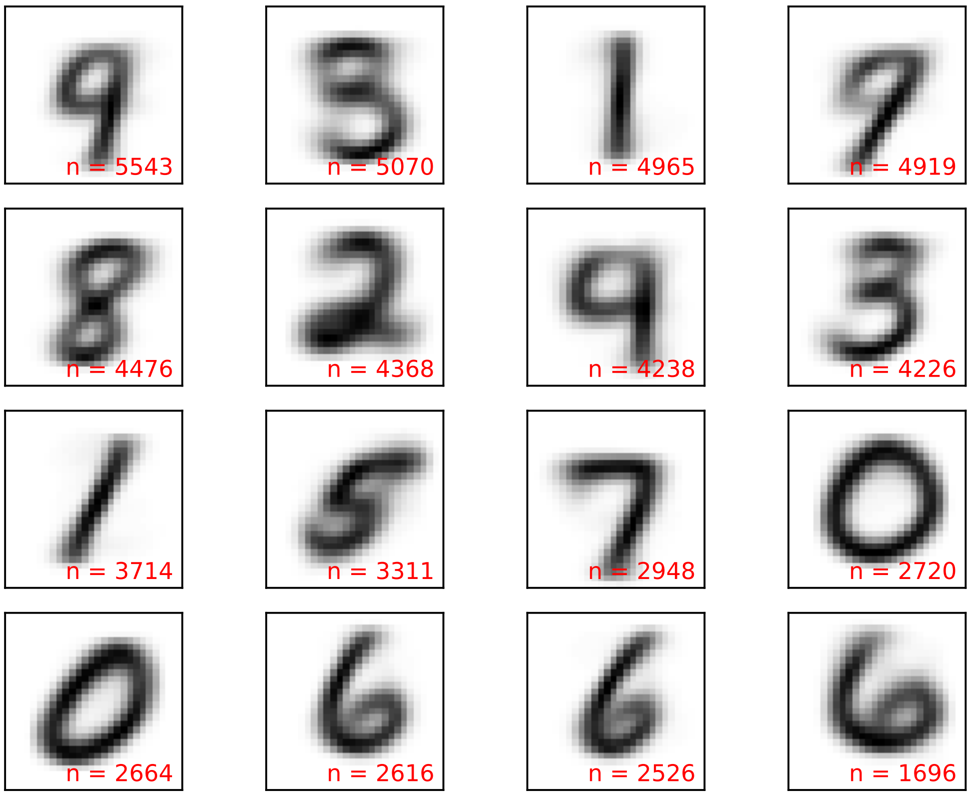

FIGURE 13. Visualization of the $k = 16$ centers orders by number of members. The number of members for each center is shown in each respective center's subplot in red in the lower-righthand corner.

5.1.2. TODO: $k = 16$ interpretation

5.2: Classification with K-means. TODO

5.3: Your thoughts? TODO