CS5785 / ORIE5750 / ECE5414 Homework 4

This homework is due on **Friday, November 3rd, 2023 at 11:59PM ET**. Upload your homework to Gradescope (Canvas->Gradescope). The homework is split into programming exercises and written exercises. Upload your homework to Gradescope. Your submission will have two parts:

1. A write-up as a single `.pdf` file. Submit this under the `hw4-report` assignment in Gradescope.
2. Source code and data files for all of your experiments (AND figures) in `.ipynb` files (file format for IPython Jupyter Notebook). These files should be placed in a folder titled `hw4` and uploaded to the `hw4-code` assignment in Gradescope.

The write-up should contain a general summary of what you did, how well your solution works, any insights you found, etc. On the cover page, include the class name, and homework number. You are responsible for submitting clear, organized answers to the questions. You could use online LaTeX templates from Overleaf, under "Homework Assignment" and "Project / Lab Report".

Please include all relevant information for a question, including text response, equations, figures, graphs, output, etc. If you include graphs, be sure to include the source code that generated them. Please pay attention to Canvas for announcements, policy changes, etc. and Piazza for homework related questions.

If you need help

There are several strategies available to you.

- If you get stuck, we encourage you to post a question on Piazza. That way, your solutions will be available to other students in the class.
- The professor and TAs offer office hours, which are a great way to get some one-on-one help.
- You are allowed to use well known libraries such as `scikit-learn`, `scikit-image`, `numpy`, `scipy`, etc. for this assignment (including implementations of machine learning algorithms), unless we explicitly say that you cannot in a particular question. Any reference or copy of public code repositories should be properly cited in your submission (examples include Github, Wikipedia, Blogs).

Programming Exercises

1. **Eigenface for face recognition. (40 pts)**

   In this assignment you will implement the Eigenface method for recognizing human faces. You will use face images from *The Yale Face Database B*, where there are 64 images under different lighting conditions per each of 10 distinct subjects, 640 face images in total.

   Read more (optional):

   - Eigenface on Wikipedia: https://en.wikipedia.org/wiki/Eigenface
   - Eigenface on Scholarpedia: http://www.scholarpedia.org/article/Eigenfaces

   (a) **(2 pts)** Download The Face Dataset and unzip `faces.zip`, You will find a folder called *images* which contains all the training and test images; *train.txt* and *test.txt* specifies the training set and test (validation) set split respectively, each line gives an image path and the corresponding label.

   (b) **(2 pts)** Load the training set into a matrix $\mathbf{X}$: there are 540 training images in total, each has 50×50 pixels that need to be concatenated into a 2500-dimensional vector. So the size of $\mathbf{X}$ should be 540×2500, where each row is a flattened face image. Pick a face image from $\mathbf{X}$ and display that image in grayscale. Do the same thing for the test set. The size of matrix $\mathbf{X_{test}}$ for the test set should be 100×2500.

   Below is the sample code for loading data from the training set. You can directly run it in Jupyter Notebook:

```
import numpy as np
from scipy import misc
from matplotlib import pylab as plt
import matplotlib.cm as cm
%matplotlib inline

train_labels, train_data = [], []
for line in open('./faces/train.txt'):
    im = misc.imread(line.strip().split()[0])
    train_data.append(im.reshape(2500,))
    train_labels.append(line.strip().split()[1])
train_data, train_labels = np.array(train_data, dtype=float), np.array(train_labels, dtype=int)

print train_data.shape, train_labels.shape
plt.imshow(train_data[10, :].reshape(50,50), cmap = cm.Greys_r)
plt.show()
```

   (c) **(3 pts)** Average Face. Compute the *average face* μ from the whole training set by summing up every row in $\mathbf{X}$ then dividing by the number of faces. Display the *average face* as a grayscale image.

   (d) **(3 pts)** Mean Subtraction. Subtract average face μ from every row in $\mathbf{X}$. That is, $\mathbf{x_i} := \mathbf{x_i} - \mu$, where $\mathbf{x_i}$ is the i-th row of $\mathbf{X}$. Pick a face image after mean subtraction from the new $\mathbf{X}$ and display that image in grayscale. Do the same thing for the test set $\mathbf{X_{test}}$ using the precomputed average face μ in (c).

(e) **(10 pts)** Eigenface. Perform eigendecomposition on $\mathbf{X}^T\mathbf{X} = \mathbf{V}\Lambda\mathbf{V}^T$ to get eigenvectors $\mathbf{V}^T$, where each row of $\mathbf{V}^T$ has the same dimension as the face image. We refer to $\mathbf{v_i}$, the i-th row of $\mathbf{V}^T$, as i-th *eigenface*. Display the first 10 eigenfaces as 10 images in grayscale.

(f) **(10 pts)** Eigenface Feature. The top r eigenfaces $\mathbf{V}^T[:r,:] = \{v_1, v_2, \ldots, v_r\}^T$ span an r-dimensional linear subspace of the original image space called *face space*, whose origin is the average face μ, and whose axes are the eigenfaces $\{v_1, v_2, \ldots, v_r\}$. Therefore, using the top r eigenfaces $\{v_1, v_2, \ldots, v_r\}$, we can represent a 2500-dimensional face image $\mathbf{z}$ as an r-dimensional feature vector $\mathbf{f}$: $\mathbf{f} = \mathbf{V}^T[:r,:]\ \mathbf{z} = [v_1, v_2, \ldots, v_r]^T\mathbf{z}$. Write a function to generate r-dimensional feature matrix $\mathbf{F}$ and $\mathbf{F_{test}}$ for training images $\mathbf{X}$ and test images $\mathbf{X_{test}}$, respectively (to get $\mathbf{F}$, multiply $\mathbf{X}$ to the transpose of first r rows of $\mathbf{V}^T$, $\mathbf{F}$ should have same number of rows as $\mathbf{X}$ and r columns; similarly for $\mathbf{X_{test}}$).

(g) **(10 pts)** Face Recognition. For this problem, you are welcome to use libraries such as `scikit learn` to perform logistic regression. Extract training and test features for $r = 10$. Train a Logistic Regression model using $\mathbf{F}$ and test on $\mathbf{F_{test}}$. Report the classification accuracy on the test set. Plot the classification accuracy on the test set as a function of r when $r = 1, 2, \ldots, 200$. Use "one-vs-rest" logistic regression, where a classifier is trained for each possible output label. Each classifier is trained on faces with that label as positive data and all faces with other labels as negative data. `sklearn` calls this the "ovr" mode.

WRITTEN EXERCISES

1. **SVD and eigendecomposition. (10 pts)** Recall that the SVD of an $m \times n$ matrix X is the factorization of X into three matrices $X = UDV^T$, where U is a $m \times m$ orthonormal matrix, D is a $m \times n$ diagonal matrix with non-negative real numbers on the diagonal, and V is a $n \times n$ orthonormal matrix. An orthonormal matrix just means that $U^TU = I$ and $V^TV = I$.

   Show that we can obtain the eigendecomposition of X^TX from the SVD of a matrix X.

   (This tells us that we can do an SVD of X and get same result as the eigendecomposition of X^TX but the SVD is faster and easier.)

2. **SVD of Rank Deficient Matrix. (10 pts)** You can use NumPy library for this problem. In this extreme example of dimensionality reduction, we are going to see the original data's dimensionality is redundant and can be completely captured by lower dimensions. Consider matrix M. It has rank 2, as you can see by observing that there times the first column minus the other two columns is 0.

$$M = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix}. \tag{1}$$

   (a) Compute the matrices M^TM and MM^T.

   (b) Find the eigenvalues for your matrices of part (a).

   (c) Find the eigenvectors for the matrices of part (a).

(d) Find the SVD for the original matrix M from parts (b) and (c). Note that there are only two nonzero eigenvalues, so your matrix Σ should have only two singular values, while U and V have only two columns.

(e) There are 2 non-zero singular values, if we only keep one by setting the smaller singular value to 0 , then the data will be represented in 1D only. Compute such one-dimensional approximation to M.