

---

# CS5785 Homework 3

---

The homework is generally split into programming exercises and written exercises.

This homework is due on **October 16, 2019 at 11:59 PM EST**. Upload your homework to [CMS](#). Please upload the submission as a single .zip file. A complete submission should include:

1. A write-up as a single .pdf file, or as a single .ipynb file.
2. Source code for all of your experiments (AND figures) in .py files if you use Python or .ipynb files if you use the IPython Notebook. If you use some other language, include all build scripts necessary to build and run your project along with instructions on how to compile and run your code.

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, homework number, and team member names. You are responsible for submitting clear, organized answers to the questions. Note that if you submit the writeup portion as a .ipynb file it should still be a clean and organized report, as with a .pdf submission, with only the minimal amount of code needed to generate figures etc. (note that you can import important functions defined in a separate .py file into a IPython notebook). If you submit as PDF could use online  $\text{\LaTeX}$  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 Slack for relevant information regarding updates, tips, and policy changes. You are encouraged (but not required) to work in groups of 2.

## IF YOU NEED HELP

There are several strategies available to you.

- If you ever get stuck, the best way is to ask on Slack. That way, your solutions will be available to the other students in the class.
- Your instructor and TAs will offer office hours<sup>1</sup>, 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. in this assignment. Any reference or copy of public code repositories should be properly cited in your submission (examples include Github, Wikipedia, Blogs).

---

<sup>1</sup><https://cs5785-2019.github.io/index.html>

## PROGRAMMING EXERCISES

### 1. Eigenface for face recognition.



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. With your implementation, you will explore the power of the Singular Value Decomposition (SVD) in representing face images.

#### Read more (optional):

- Eigenface on Wikipedia: <https://en.wikipedia.org/wiki/Eigenface>
  - Eigenface on Scholarpedia: <http://www.scholarpedia.org/article/Eigenfaces>
- (a) Download [The Face Dataset](#). After you 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) Load the training set into a matrix  $\mathbf{X}$ : there are 540 training images in total, each has  $50 \times 50$  pixels that need to be concatenated into a 2500-dimensional vector. So the size of  $\mathbf{X}$  should be  $540 \times 2500$ , 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  $\mathbf{X}_{\text{test}}$  for the test set should be  $100 \times 2500$ .

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

---

```

1  import numpy as np
2  from scipy import misc
3  from matplotlib import pylab as plt
4  import matplotlib.cm as cm
5  %matplotlib inline
6
7  train_labels, train_data = [], []
8  for line in open('./faces/train.txt'):
9      im = misc.imread(line.strip().split()[0])
10     train_data.append(im.reshape(2500,))
11     train_labels.append(line.strip().split()[1])
12  train_data, train_labels = np.array(train_data, dtype=float), np.array(train_labels, dtype=int)
13
14  print train_data.shape, train_labels.shape
15  plt.imshow(train_data[10, :].reshape(50,50), cmap = cm.Greys_r)
16  plt.show()

```

---

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

- (d) **Mean Subtraction.** Subtract average face  $\mu$  from every column in  $\mathbf{X}$ . That is,  $\mathbf{x}_i := \mathbf{x}_i - \mu$ , where  $\mathbf{x}_i$  is the  $i$ -th column 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  $\mathbf{X}_{\text{test}}$  using the pre-computed average face  $\mu$  in (c).
- (e) **Eigenface.** Perform Singular Value Decomposition (SVD) on training set  $\mathbf{X}$  ( $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ ) to get matrix  $\mathbf{V}^T$ , 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) **Low-rank Approximation.** Since  $\Sigma$  is a diagonal matrix with non-negative real numbers on the diagonal in non-ascending order, we can use the first  $r$  elements in  $\Sigma$  together with first  $r$  columns in  $\mathbf{U}$  and first  $r$  rows in  $\mathbf{V}^T$  to approximate  $\mathbf{X}$ . That is, we can approximate  $\mathbf{X}$  by  $\hat{\mathbf{X}}_r = \mathbf{U}[:, :r] \Sigma[:, :r] \mathbf{V}^T[:, :r]$ . The matrix  $\hat{\mathbf{X}}_r$  is called rank- $r$  approximation of  $\mathbf{X}$ . Plot the rank- $r$  approximation error  $\|\mathbf{X} - \hat{\mathbf{X}}_r\|_F$ <sup>2</sup> as a function of  $r$  when  $r = 1, 2, \dots, 200$ .
- (g) **Eigenface Feature.** The top  $r$  eigenfaces  $\mathbf{V}^T[:, :r] = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r\}^T$  span an  $r$ -dimensional linear subspace of the original image space called *face space*, whose origin is the average face  $\mu$ , and whose axes are the eigenfaces  $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r\}$ . Therefore, using the top  $r$  eigenfaces  $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r\}$ , we can represent a 2500-dimensional face image  $\mathbf{z}$  as an  $r$ -dimensional feature vector  $\mathbf{f}$ :  $\mathbf{f} = \mathbf{V}^T[:, :r] \mathbf{z} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r]^T \mathbf{z}$ . Write a function to generate  $r$ -dimensional feature matrix  $\mathbf{F}$  and  $\mathbf{F}_{\text{test}}$  for training images  $\mathbf{X}$  and test images  $\mathbf{X}_{\text{test}}$ , 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  $\mathbf{X}_{\text{test}}$ ).
- (h) **Face Recognition.** Extract training and test features for  $r = 10$ . Train a Logistic Regression model using  $\mathbf{F}$  and test on  $\mathbf{F}_{\text{test}}$ . Report the classification accuracy on the test set. Plot the classification accuracy on the test set as a function of  $r$  when  $r = 1, 2, \dots, 200$ . 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 “ovr” mode.
- 2. Clustering for text analysis.** In this problem, you will analyze all the articles from the journal Science in the year 2000. (Thanks to JSTOR for providing the data.) Many of the parameters of this analysis will be left for you to decide. For these files you will need to use `science2k-vocab.npy` and `science2k-titles.npy`, which are vectors of terms and titles respectively.

- (a) The file `science2k-doc-word.npy` contains a  $1373 \times 5476$  matrix, where each row is an article in Science described by 5476 word features. The articles and words are in the same order as in the vocabulary and titles files above. You can read this file using

```
numpy.load("science2k-doc-word.npy")
```

To obtain the features, we performed the following transformation. First, we computed per-document smoothed word frequencies. Second, we took the log of those frequencies. Finally, we centered the per-document log frequencies to have zero mean.

Cluster the documents using  $k$ -means and various values of  $k$  (go up to at least  $k = 20$ ).

Select a value of  $k$ .

---

<sup>2</sup> $\|\cdot\|_F$  is the **Frobenius Norm** of a matrix:  $\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2}$ , which can be directly computed in `numpy`.

For that value, report the top 10 words of each cluster in order of the largest positive distance from the average value across all data. More specifically, if  $\bar{\mathbf{x}}$  is the 5476-vector of average values across documents and  $\mathbf{m}_i$  is the  $i$ th mean, report the words associated with the top components in  $\mathbf{m}_i - \bar{\mathbf{x}}$ . Report the top ten documents that fall closest to each cluster center. You can find the titles in the `science2k-titles.dat` file.

Comment on these results. What has the algorithm captured? How might such an algorithm be useful?

- (b) The file `science2k-word-doc.txt` is similar, but capture *term-wise* rather than document-wise features. That is, for each *term*, we count the frequency as the number of *documents* that term appears in rather than the other way around. This allows us to characterize individual terms.

This matrix is  $5476 \times 1373$ , where each row is a term in Science described by 1373 “document” features. These are transformed document frequencies (as above). Repeat the analysis above, but cluster terms instead of documents. The terms are listed in `science2k-vocab.txt`

Comment on these results. How might such an algorithm be useful? What is different about clustering terms from clustering documents?

### 3. EM algorithm and implementation

- (a) The parameters of Gaussian Mixture Model (GMM) can be estimated via the EM algorithm. Show that the alternating algorithm for  $k$ -means (in Lec. 11) is a special case of the EM algorithm and show the corresponding objective functions for E-step and M-step.
- (b) Download the [Old Faithful Geyser Dataset](#). The data file contains 272 observations of (*eruption time, waiting time*). Treat each entry as a 2 dimensional feature vector. Parse and plot all data points on 2-D plane.
- (c) Implement a bimodal GMM model to fit all data points using EM algorithm. Explain the reasoning behind your termination criteria. For this problem, we assume the covariance matrix is spherical (i.e., it has the form of  $\sigma^2 I$  for scalar  $\sigma$ ) and you can randomly initialize Gaussian parameters. For evaluation purposes, please submit the following figures:
- Plot the trajectories of two mean vectors in 2 dimensions (i.e., coordinates vs. iteration).
  - Run your program for 50 times with different initial parameter guesses. Show the distribution of the total number of iterations needed for algorithm to converge.
- (d) Repeat the task in (c) but with the initial guesses of the parameters generated from the following process:
- Run a  $k$ -means algorithm over all the data points with  $K = 2$  and label each point with one of the two clusters.
  - Estimate the first guess of the mean and covariance matrices using maximum likelihood over the labeled data points.

Compare the algorithm performances of (c) and (d).

## WRITTEN EXERCISES

1. **SVD of Rank Deficient Matrix.** Consider matrix  $M$ . It has rank 2, as you can see by observing that three times the first column minus the other two columns is 0.

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

- (a) Compute the matrices  $M^T M$  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  $\Sigma$  should have only two singular values, while  $U$  and  $V$  have only two columns.
  - (e) Set your smaller singular value to 0 and compute the one-dimensional approximation to the matrix  $M$ .
2. **Principal Components of Standardized Data.** Recall that the principal components of uncentered data  $\mathbf{X}$  are given by the eigenvectors of  $(\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}})$  in descending order of corresponding eigenvalue, where  $\bar{\mathbf{x}}$  is the mean of the rows of  $\mathbf{X}$ . The matrix  $\hat{\Sigma} = \frac{1}{n} (\mathbf{X} - \bar{\mathbf{x}})^T (\mathbf{X} - \bar{\mathbf{x}})$  is known as the *covariance* matrix of  $\mathbf{X}$  and it is always PSD. Suppose it is also invertible. Let  $\mathbf{X}^S = (\mathbf{X} - \bar{\mathbf{x}}) \hat{\Sigma}^{-1/2}$  be the standardization of  $\mathbf{X}$  – that is, a linear transformation of  $\mathbf{X}$  so that each column has zero mean and unit variance and each two columns have zero covariance. Such standardization is a common preprocessing used to make any point cloud  $\mathbf{X}$  look like a symmetric sphere around the origin. What are the principal components of the standardized data  $\mathbf{X}^S$ ? Explain why we get the result we get in terms of PCA finding the directions of largest variation and comment on the practical implications of this for PCA applied to data where each column may be in different units of scale.