
CS5785 Homework 3

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

This homework is due on **October 22, 2025 at 11:59 PM EST**. Upload your homework to [Gradescope](#). There are two assignments for this homework in Gradescope. Please note a complete submission should include:

1. A write-up as a single .pdf file, which should be submitted to “Homework 2 (write-up)” This file should contain your answers to the written questions **and** exported pdf file / structured write-up of your answers to the coding questions (which should include core codes, plots, outputs, and any comments / explanations).
2. Source code for all of your experiments (AND figures) zipped into a single .zip file, 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. **If you use the IPython Notebook to create any graphs, please make sure you also include them in your write-up.** This should be submitted to “Homework 2 (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. 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. You are encouraged (but not required) to work in groups of 2.

Lastly, if you use generative AI (e.g., ChatGPT) as a tool for any of the problems, you are required to include a statement that describes how you used that tool. You are not allowed to copy-paste output from generative AI systems directly: you must always use your own words.

IF YOU NEED HELP

There are several strategies available to you.

- If you get stuck, we encourage you to post a question on the Discussions section of Canvas. That way, your questions/solutions will be available to other students in the class.
- Your instructor and TAs will 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. in this assignment. 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. (45 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:

```

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) **(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 pre-computed average face μ in (c).

- (e) **(10 pts)** Eigenface. Perform eigendecomposition on $\mathbf{X}^T \mathbf{X} = \mathbf{V} \mathbf{\Lambda} \mathbf{V}^T$ 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 $\mathbf{V}^T[:r,:] = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r\}^T$ span an r -dimensional linear subspace of the original image space called *face space*, whose origin is the average face μ , 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}_{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 $r = 1, 2, \dots, 200$. 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 the “ovr” mode.
- (h) **(5 pts)** Low-Rank Data Loss. These feature matrices \mathbf{F} can be mapped back to their original dimensions; however, information is lost in the process and only approximations \mathbf{X}' will be recovered. This is done by multiplying the feature matrices \mathbf{F} once again by the first r rows of \mathbf{V}^T . Plot the average Frobenius distance $d(\mathbf{X}, \mathbf{X}') = \sqrt{\text{tr}((\mathbf{X} - \mathbf{X}')^T (\mathbf{X} - \mathbf{X}'))}$ between the original data \mathbf{X} and their approximations \mathbf{X}' , over $r = 1, 2, \dots, 200$.
2. **Implement EM algorithm. (40 pts)** In this problem, you will implement a bimodal GMM model fit using the EM algorithm. Bimodal means that the distribution has two peaks, or that the data is a mixture of two groups. If you want, you can assume the covariance matrix is diagonal (i.e. it has the form $\text{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_d^2)$ for scalars σ_i) and you can randomly initialize the parameters of the model.

You will need to use the [Old Faithful Geyser Dataset](#). The data file contains 272 observations of the waiting time between eruptions and the duration of each eruption for the Old Faithful geyser in Yellowstone National Park.

You should do this without calling the [Gaussian Mixture](#) library in `scikit learn`. You can use `numpy` or `scipy` for matrix calculation or generating Gaussian distributions.

- (a) **(2 pts)** Treat each data entry as a 2 dimensional feature vector. Parse and plot all data points on 2-D plane.
- (b) **(3 pts)** Recall that EM learns the parameter θ of a Gaussian mixture model $P_\theta(x, z)$ over a dataset $D = \{x^{(i)} | i = 1, 2, \dots, n\}$ by performing the E-step and the M-step for $t = 0, 1, 2, \dots$. We repeat the E-step and M-step until convergence.

In the E-step, for each $x^{(i)} \in D$, we compute a vector of probabilities $P_{\theta_t}(z = k | x)$ for the event that each $x^{(i)}$ originates from a cluster k given the current set of parameters θ_t .

Write the expression for $P_{\theta_t}(z = k | x)$, which is the posterior of each data point $x^{(i)}$. Recall

that by Bayes' rule,

$$P_{\theta_t}(z = k | x) = \frac{P_{\theta_t}(z = k, x)}{P_{\theta_t}(x)} = \frac{P_{\theta_t}(z = k, x)}{\sum_{l=1}^K P_{\theta_t}(x|z = l)P_{\theta_t}(z = l)}.$$

Note that we have seen this formula in class. We are asking you to write it down and try to understand and it before implementing it in part (e).

- (c) **(5 pts)** In the M-step, we compute new parameters θ_{t+1} . Our goal is to find μ_k, Σ_k and ϕ_k that optimize

$$\max_{\theta} \left(\sum_{k=1}^K \sum_{x \in D} P_{\theta_t}(z_k | x) \log P_{\theta}(x | z_k) + \sum_{k=1}^K \sum_{x \in D} P_{\theta_t}(z_k | x) \log P_{\theta}(z_k) \right)$$

Write down the formula for μ_k, Σ_k , and for the parameters ϕ at the M-step (we have also seen these formulas in class).

- (d) **(25 pts)** Implement and run the EM algorithm. Specifically:
- (10 pts)** Implement the EM algorithm from scratch (e.g., in Python and numpy).
 - (5 pts)** Choose a termination criterion for when the algorithm stops repeating the E-step and the M-step. State your termination criterion and explain the reasoning behind it.
 - (10 pts)** Plot the trajectories of the two mean vectors (μ_1 and μ_2) in two dimensions as they change over the course of running EM. You might want to use a scatter plot for this.
- (e) **(5 pts)** If you were to run K -means clustering instead of the EM algorithm you just implemented, do you think you will get different clusters? You are welcome to experiment with K -means clustering on the same dataset with $K = 2$. (The [KNN](#) library from `scikit learn` is a good way to try). Comment on why do you think the results will or will not change. .

WRITTEN EXERCISES

1. Principal Component Analysis. (25 pts)

In this exercise, you will familiarize yourself with the practical steps of PCA. Given the following five 3D data points:

$$\begin{aligned} \mathbf{x}_1 &= (0, 0, 0), \\ \mathbf{x}_2 &= (1, 10, -1), \\ \mathbf{x}_3 &= (-1, 10, 1), \\ \mathbf{x}_4 &= (1, -10, -1), \\ \mathbf{x}_5 &= (-1, -10, 1), \end{aligned}$$

Your task is to perform dimensionality reduction by projecting this data onto the 2D subspace that captures the most significant variance.

- (a) **(10 pts)** Find the first, second, and third principal components of the five 3D data points and report the matrix W . **Hint:** *You can just plot the data points and think about the alignment*

of data points visually. It's possible to obtain the principal components without manually performing eigendecomposition so you should not need to do that. Also, if a vector is a principal component, its negation is also a principal component, so feel free to provide just one of them.

- (b) **(10 pts)** Project the data points onto the first two principal components to obtain the two-dimensional representation. Write down the projected vectors $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4, \mathbf{z}_5 \in \mathbb{R}^2$.
 - (c) **(5 pts)** What is the numerical variance of the data points along the third principal component? What does this variance value tell us about the data structure?
2. **Local Optima in K-means. (5pts)** Depending on initialization, K-means can get stuck on local optima. Construct an example dataset (>3 points), and provide two different initializations for it such that K-means would return different final clusterings on it.
3. **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 $U^T U = I$ and $V^T V = I$.

Show that we can obtain the eigendecomposition of $X^T X$ 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 $X^T X$ but the SVD is faster and easier, because X is a smaller matrix than $X^T X$)