EECS545 Machine Learning
Homework #5

Due date: 11:55pm, Tue 3/29/2022

Reminder: This homework is mainly about unsupervised learning. This time you will work with the **py** files that we provided for programming questions. You will be asked to fill in the functions in the given python files instead of creating new one. While you are encouraged to discuss problems in a small group (up to 5 people), you should write your own solutions and source codes independently. In case you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/umich/winter2022/eecs545wn2022/home` with a reference to the specific question in the subject line (E.g. RE: Homework 5, Q1(b)).

Submission Instructions: You should submit both your **solution** and **source code**. We may inspect your source code submission visually and run the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit your **solution** to **Gradescope**
  - Your solution should contain your answers to **all** questions (typed or hand-written).
  - Your solution should be self-contained and self-explanatory, regardless of the source code submission.

Source Code Instructions:

- Submit your **source code** and **Q5 output files** to the **Gradescope code submission**.
  - Source code should contain your code to programming questions.
  - Your submission to gradescope should contain all your code `q1.py`, `q2.py`, `q3.py`, and `q5.py`. Do not use a different name for the code or the autograder may not find your submission.
  - To receive full credit your code must run and terminate without error in the autograder. Your code must output all important information to **stdout**.
  - Your submission should also contain `q5_unmixed_track_X.wav`, for `X=0...4`, which is generated by `q5.py`. We will grade this portion by downloading your uploaded sound files.

Your source code should run under an environment with following libraries:

- Python 3.6+
- Numpy, Scipy, Pytorch, and Sklearn (for implementations of algorithms)
- Matplotlib (for plots)
- imageio (for image loading)

Please do not use any other library unless otherwise instructed. You should be able to load the data and execute your code on someone else's computer with the environment described above. In other words, work with the setup we provide and do not change the directory structure. Use relative path instead of absolute path to load the data. **Note**, the output of your source code must match with your solution pdf.

1 [15 + 2 points] K-means for image compression

In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image. `q12data` directory contains `mandrill-large.tiff` and `mandrill-small.tiff`. `mandrill-large.tiff` is a 512×512 image of a mandrill represented in 24-bit color. This means that, for each of the $512 \times 512 = 262,144$ pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262,144 \times 3 = 786,432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to $K = 16$ colors. More specifically, each pixel in the image is considered as a point in the three-dimensional (r, g, b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid. Follow the instructions below.

(a) (10 pts) The starter code `q1.py` reads an image `mandrill-small.tiff`. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, implement and run K-means with 16 clusters on the pixel data from this image, iterating 50 times. Measure and report the pixel error at each iteration. We provided the initial centroids as `initial-centroids` in the starter code. Also, for the distance between centroids and data points, use the provided `sklearn.metrics.pairwise_distances` function.

(b) (Extra 2 pts) You will get extra 2 points for vectorized implementation in the part (a). One point will be given for your own vectorized implementation of `sklearn.metrics.pairwise_distances` function. One point will be given for having only one loop corresponding to the EM iterations (no nested loop).

(c) (3 pts) After training, read the test image `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid. Display the new image, and measure the pixel error using provided `calculate_error` function.

(d) (2 pts) If we represent the image with these reduced 16 colors, by (approximately) what factor have we compressed the image (in terms of bits used to represent pixels)?

2 [15 + 2 points] Gaussian mixtures for image compression

In this problem, we will repeat the problem 1, but implement Gaussian mixtures (with full covariances) with $K = 5$ instead of K-means. We are using the same image data setting as in the problem 1.

(a) (10 pts) The starter code is provided as `q2.py`. Implement Gaussian mixtures with $K = 5$. Iterate 50 times and report the log-likelihood of the training data at each iteration. We provided the initial mean and covariance matrices, and prior distribution of latent cluster as `initial-mu`, `initial-sigma`, and `initial-pi` in the starter code.

(b) (Extra 2 pts) You will get extra 2 points for vectorized implementation in the part (a). The goal is to have one loop in E step and another in M step where both loops go over the cluster index k. One point will be given for having one loop in E step. One point will be given for having one loop in M step.

(c) (3 pts) After training, read the test image `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the value of latent cluster mean, where we use the MAP estimation for the latent cluster-assignment variable for each pixel. Display the new image, and measure the pixel error using provided `calculate_error` function. In addition, report the model parameters $\{(\mu_k, \Sigma_k) : k = 1, ..., 5\}$.

(d) (2 pts) If we represent the image with these reduced 5 colors, by (approximately) what factor have we compressed the image?

3 [25 points] PCA and eigenfaces

(a) (10 pts) In lecture, we derived PCA from the "maximizing variance" viewpoint. In this problem, we will take the "minimizing squared error" viewpoint. Assume the data is zero-centered. Let $K \in \{0, 1, \ldots, d\}$ be arbitrary and let $\mathbf{x}^{(n)} \in \mathbb{R}^d$. Let $\mathcal{U} = \{\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_K] \in \mathbb{R}^{d\times K} | \{\mathbf{u}_i\}_{i=1}^K\text{'s are orthonormal vectors}\}$, where $\mathbf{u}_i$ is the i-th column vector of $\mathbf{U}$. Show that the following equation holds

$$\min_{\mathbf{U}\in\mathcal{U}} \frac{1}{N}\sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \mathbf{U}\mathbf{U}^T\mathbf{x}^{(n)} \right\|^2 = \min_{\mathbf{U}\in\mathcal{U}} \frac{1}{N}\sum_{n=1}^{N} \left\| \mathbf{x}^{(n)} - \sum_{i=1}^{K} \mathbf{u}_i\mathbf{u}_i^T\mathbf{x}^{(n)} \right\|^2 = \sum_{k=K+1}^{d} \lambda_k,$$

where $\lambda_1 \geq \ldots \geq \lambda_d$ are the (ordered) eigenvalues of the data covariance $\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}^{(n)} - \bar{\mathbf{x}})(\mathbf{x}^{(n)} - \bar{\mathbf{x}})^T$, $\bar{\mathbf{x}}$ is the data mean vector. Also, show that the $\mathbf{u}_i$'s that solves the minimization problem above is the eigenvectors corresponding to the (ordered) eigenvalues λ_i's. Here, $\mathbf{U}\mathbf{U}^T\mathbf{x}^{(n)}$ is called a projection of $\mathbf{x}^{(n)}$ into the subspace spanned by $\mathbf{u}_i$'s.

Now, you will apply PCA to face images. The principal components (eigenvectors) of the face images are called eigenfaces.

(b) (6 pts) Use the starter code q3.py to load the data from q3.npy. By regarding each image as a vector in a high dimensional space, perform PCA on the face images (sort the eigenvalues in descending order). Report the eigenvalues corresponding to the first 10 principal components. Plot **all** the eigenvalues (in sorted order) where x-axis is the index of corresponding principal components and y-axis is the eigenvalue.

(c) (5 pts) Hand in a 2×5 array of subplots showing the first 10 principal components/eigenvectors ("eigenfaces") (sorted according to the descending eigenvalues) as images, treating the mean of images as the first principal component. Comment on what facial or lighting variations some of the different principal components are capturing (Note: you don't need to comment for all the images. Just pick a few that capture some salient aspects of image).

(d) (4 pts) Eigenfaces are a set of bases for all the images in the dataset (every image can be represented as a linear combination of these eigenfaces). Suppose we have L eigenfaces in total. Then we can use the L coefficients (of the bases, i.e. the eigenfaces) to represent an image. Moreover, we can use the first $K(< L)$ eigenfaces to reconstruct the face image approximately (correspondingly use K coefficients to represent the image). In this case, we reduce the dimension of the representation of images from L to K. To determine the proper K to use, we will check the percentage of variance that has been preserved (recall that the basic idea of PCA is preserving variance). Specifically, we define total variance

$$v(K) = \sum_{i=1}^{K} \lambda_i$$

where $1 \leq K \leq L$ and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_L$ are eigenvalues. Then the percentage of total variance is

$$\frac{v(K)}{v(L)}$$

How many principal components are needed to represent 95% of the total variance? How about 99%? What is the percentage of reduction in dimension in each case?

4 [25 + 2 points] Expectation Maximization

Consider a probabilistic model with random variables $z \in \{0, 1\}, \epsilon \in \mathbb{R}$, and $x \in \mathbb{R}$. The random variables z and ϵ are independent. The joint distribution of all three variables is as follows:

$$\begin{aligned} z &\sim \text{Bernoulli}(\phi) \\ \epsilon &\sim \text{Normal}(\mu, \sigma^2) \\ x &= z + \epsilon \end{aligned}$$

The parameters of this model are $\phi \in [0, 1], \mu \in \mathbb{R}$, and $\sigma \in \mathbb{R}$.

(a) (2 pts) Write down in English a simple, one-line description of what the marginal distribution of x looks like.

(b) (8 pts) Suppose we have a training set $\{(z^{(1)}, \epsilon^{(1)}, x^{(1)}), \ldots, (z^{(N)}, \epsilon^{(N)}, x^{(N)})\}$, where all three variables z, ϵ, x are observed. Write down the log-likelihood of the variables, and derive the maximum likelihood estimates of the model's parameters.

(c) (15 pts) Now, suppose z and ϵ are latent (unobserved) random variables. Our training set is therefore of the form $\{x^{(i)}, \ldots, x^{(N)}\}$. Write down the log-likelihood of the variables, and derive an EM algorithm to maximize the log-likelihood. Clearly indicate what are the E-step and the M-step.

(d) (Extra 2 pts) Consider the modified probabilistic model in the following:

$$\begin{aligned} z &\sim \text{Bernoulli}(\phi) \\ \epsilon &\sim \text{Normal}(\mu, \sigma^2) \\ x &= \lambda z + \epsilon, \end{aligned}$$

where $\lambda \in \mathbb{R}$ is the extra parameter. Why would you consider such modification in the model? Is there any advantage of the modified model over the original model? [Hint: Which model is more general? Why?]

5 [20 points] Independent Component Analysis

In this problem, you will implement maximum-likelihood Independent Component Analysis (ICA) for blind audio separation. As we learned in the lecture, the maximum-likelihood ICA minimizes the following loss:

$$\ell(W) = \sum_{i=1}^{N} \left(\sum_{j=1}^{m} \log g'\left(w_j^T x^{(i)}\right) + \log |W| \right), \tag{1}$$

where N is the number of time steps, m is the number of independent sources, W is the transformation matrix representing a concatenation of w_j's, and $g(s) = 1/\left(1 + e^{-s}\right)$ is the sigmoid function. This link has some nice demos of blind audio separation: `https://cnl.salk.edu/~tewon/Blind/blind_audio.html`. We provided the starter code `q5.py` and the data `q5.dat`. The file `q5.dat` contains a matrix with 5 columns, with each column corresponding to a mixed signal recorded from one of the microphone.

(a) Implement, run ICA, and report the W matrix you found. The starter code will write the mixed and unmixed sources to `q5_x.wav` files. To make sure your code is correct, you should listen to the resulting unmixed sources. (Some overlap in the sources may be present, but the different sources should be pretty clearly separated.) Submit all your unmixed sound files (`q5_unmixed_track_X.wav`) along with your source code to the Gradescope code submission.