CS5785 Homework 4

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

This homework is due on **Dec 3rd, 2020 at 11:59 PM ET**. Upload your homework to Gradescope (Canvas->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. ⟶ Submit to "Homework 4- Write Up".
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.** ⟶ Submit to "Homework 4- 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" or "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 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 get stuck, we encourage you to post a question on the Discussions section of Canvas. 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. Any reference or copy of public code repositories should be properly cited in your submission (examples include Github, Wikipedia, Blogs).

PROGRAMMING EXERCISES

You will need to download extra data files for the following problems. You can find them on Canvas under Files->Homework_data.

1. **Clustering for text analysis. (20 pts)** 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) **(10 pts)** The extra data file `science2k-doc-word.npy` contains a 1373×5476 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.

   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 ith 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.txt` file.

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

   (b) **(10 pts)** The file `science2k-word-doc.npy` 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 5476×1373, 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?

2. **EM algorithm and implementation (25 pts)**

   (a) **(5 pts)** 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.

   (b) **(10 pts)** 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 σ) 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.

(c) **(10 pts)** Repeat the task in (b) 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 (b) and (c).

3. **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. 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) **(2 pts)** Download and unzip The Face Dataset (`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 $\mathbf{X}_{\text{test}}$ 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 pre-computed 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. 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 "ovr" mode.

WRITTEN EXERCISES

1. **SVD and eigendecomposition. (5 pts)** Show that from the Singular Value Decomposition (SVD) of a matrix X, we can obtain the eigendecomposition of $X^T X$. 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.

2. **Weights for clustering (ESL Exercise 14.1). (5 pts)** In clustering algorithms like K-means, we need to compute distances in the feature space. Sometimes people use weights to value some feature more than others. Show that weighted Euclidean distance for p dimensional data points x_i and $x_{i'}$

$$d_e^{(w)}(x_i, x_{i'}) = \frac{\sum_{l=1}^{p} w_l (x_{il} - x_{i'l})^2}{\sum_{l=1}^{p} w_l} \tag{1}$$

satisfies

$$d_e^{(w)}(x_i, x_{i'}) = d_e(z_i, z_{i'}) = \sum_{l=1}^{p} (z_{il} - z_{i'l})^2, \tag{2}$$

where

$$z_{il} = x_{il} \cdot \left(\frac{w_l}{\sum_{l=1}^{p} w_l} \right)^{1/2}. \tag{3}$$

Thus weighted Euclidean distance based on x is equivalent to unweighted Euclidean distance based on a proper transformed data z.

3. **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.

$$M = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix}. \tag{4}$$

   (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 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.