CS5785 Homework 3

The homework is generally split into programming exercises and written exercises.
This homework is due on **October 18, 2018 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 important functions defined in a separate `.py` file into a IPython notebook). If you submit as PDF could use online LaTeX templates from Overleaf, under "Homework Assignment" and 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[1], 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).

[1] https://cs5785-2018.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 50×50 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) Average Face. Compute the *average face* μ 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 μ 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 $\mathbf{X_{test}}$ using the pre-computed average face μ 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 Σ is a diagonal matrix with non-negative real numbers on the diagonal in non-ascending order, we can use the first r elements in Σ 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 $\hat{\mathbf{X}}_\mathbf{r} = \mathbf{U}[:,:r]\ \Sigma[:r,:r]\ \mathbf{V}^T[:r,:]$. The matrix $\hat{\mathbf{X}}_\mathbf{r}$ is called rank-r approximation of $\mathbf{X}$. Plot the rank-r approximation error $\|\mathbf{X} - \hat{\mathbf{X}}_\mathbf{r}\|_F$ [2] as a function of r when $r = 1, 2, \ldots, 200$.

(g) 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 $\mathbf{X_{test}}$).

(h) 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.

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 1373×5476 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.

[2] $\|.\|_F$ is the Frobenius Norm of a matrix: $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{i=1}^{n}|a_{ij}|^2}$, 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 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.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 5476×1373, 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 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.

(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).

4. **Multidimensional scaling for genetic population differences.** In this exercise, we will look at a dataset of 42 human geographic populations collected by Cavalli-Sforza *et al.* in *The History and Geography of Human Genes*, 1994. There are many ways to measure genetic similarity between

populations. This work uses the *modified Nei's distance,* which compares allele frequencies at specific locations within the human genome. Other ways to measure genetic similarity include the edit distance between DNA sequences or Euclidean distance of gene expressions.

This dataset contains comparisons between every pair of populations in the study as a distance matrix. You can use the following code to interact with it:

```
import numpy as np
data = np.load("mds-population.npz")
print data['D'] # Distance matrix
print data['population_list'] # List of populations
```

Here, $D_{i,j}$ is the dissimilarity between population i and j. Higher scores indicate more dissimilarity.

Since this distance is not based on an underlying vector representation, we cannot directly use traditional techniques like classification or clustering to analyze it. Our first step will be to *embed* this distance matrix into an m-dimensional vector space.

(a) First, use multidimensional scaling (MDS) to coerce D into a 2-dimensional vector representation. You can use the functions in `sklearn.manifold` for this.

   i. MDS will attempt to output a set of points $\mathbf{x} \in \mathscr{R}^{42,m}$ such that the Euclidean distance between every pair of points approximates the Nei's distance between these populations, or $\|x_i - x_i\|_2 \approx D_{i,j}$. What assumptions are being made? Under what circumstances could this fail? How could we measure how much information is being lost? Please explain.

   ii. One way of increasing the quality of the output is by increasing the dimensionality of the MDS result. How many dimensions are necessary to capture most of the variation in the data? There are several ways of making this judgment: for instance, you might sample some quality measure between x and D while varying m, or you might count the nonzero singular values of D, or inspect the singular values of x at some high dimension like $m = 20$. Briefly explain your method and justify why it makes sense.

   iii. Use MDS to embed the distance matrix into only two dimensions and show the resulting scatterplot. Label each point with the name of its population.

(b) *k-means on 2D embedding.* Select an appropriate k and run k-means on the scatterplot. Show the resulting clusters.

   Since we are working with only two dimensions, this clustering is likely to lose a lot of high-dimensional structure. Do you agree with the resulting clustering? What information seems to be lost?

(c) *Comparing hierarchical clustering with K-Means.* Use hierarchical clustering to cluster the original distance matrix. We suggest using the functions in `scipy.cluster.hierarchy` for this, as this library can plot the resulting graph structure. Show the resulting tree as a *dendrogram*, labeling the x axis with the categorical population names and the y axis with the Nei's distance between clusters. (It's also possible to do this with `sklearn`, but traversing the resulting structure is much harder).

   To turn the resulting tree into a flat clustering of points, cut off the dendrogram at a certain distance by merging all subclusters within this distance together. This can be done with the

`scipy.cluster.hierarchy.fcluster` function. Select a distance cutoff that roughly corresponds with your chosen k earlier, balancing the number of points in each cluster with the number of resulting clusters. Visualize the resulting clustering by coloring the corresponding points on your 2D MDS embedding. How does this clustering compare with the k-means clustering you computed earlier?

(d) *Compare k-medoids with k-means.* Repeat the above experiment, but applying k-medoid clustering on the original distance matrix. Show the resulting clusters on a 2D scatterplot. Are there any significant differences between the clustering chosen by k-medoids compared to k-means?

WRITTEN EXERCISES

1. **SVD of Rank Deficient Matrix.** 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 $M^T M$ 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) 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 $(\mathbf{X}-\bar{x})^T(\mathbf{X}-\bar{x})$ in descending order of corresponding eigenvalue, where $\bar{x}$ is the mean of the rows of $\mathbf{X}$. The matrix $\hat{\Sigma} = \frac{1}{n}(\mathbf{X}-\bar{x})^T(\mathbf{X}-\bar{x})$ is known as the *covariance* matrix of $\mathbf{X}$ and it is always PSD. Suppose it is also invertible. Let $\mathbf{X}^S = (\mathbf{X}-\bar{x})\hat{\Sigma}^{-1/2}$ 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.