CS5785 Homework 2

The homework is generally split into programming exercises and written exercises.
This homework is due on **October 10, 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
2. Source code for all of your experiments (AND figures) in `.py` files if you use Python or `.ipynb` files if you use the Jupyter 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. You 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://cornelltech.github.io/cs5785-fall-2019/contact.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 $\mathbf{X}_{\mathbf{test}}$ 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 row 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 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).

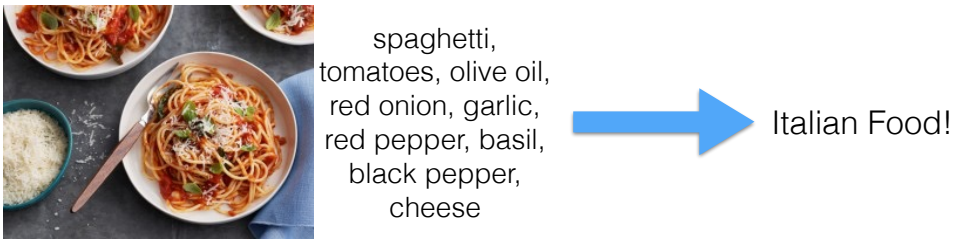
(e) Eigenface. Perform Singular Value Decomposition (SVD) on training set $\mathbf{X}$ ($\mathbf{X} = \mathbf{U\Sigma V}^T$) to get matrix $\mathbf{V}^T$, where each row of $\mathbf{V}^T$ has the same dimension as the face image. We refer to $\mathbf{v_i}$, the i-th row of $\mathbf{V}^T$, as i-th *eigenface*. 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. What's Cooking?



(a) Join the What's Cooking competition on Kaggle. Download the training and test data (in `.json`). The competition page describes how these files are formatted.

(b) Tell us about the data. How many samples (dishes) are there in the training set? How many categories (types of cuisine)? Use a list to keep all the unique ingredients appearing in the training set. How many unique ingredients are there?

[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.

(c) Represent each dish by a binary ingredient feature vector. Suppose there are d different ingredients in total from the training set, represent each dish by a $1 \times d$ binary ingredient vector $\mathbf{x}$, where $\mathbf{x_i} = 1$ if the dish contains ingredient i and $\mathbf{x_i} = 0$ otherwise. For example, suppose all the ingredients we have in the training set are { beef, chicken, egg, lettuce, tomato, rice } and the dish is made by ingredients { chicken, lettuce, tomato, rice }, then the dish could be represented by a 6×1 binary vector $[0,1,0,1,1,1]$ as its feature or attribute. Use $n \times d$ feature matrix to represent all the dishes in training set and test set, where n is the number of dishes.

(d) Using Naïve Bayes Classifier to perform 3 fold cross-validation on the training set and report your average classification accuracy. Try both Gaussian distribution prior assumption and Bernoulli distribution prior assumption.

(e) For Gaussian prior and Bernoulli prior, which performs better in terms of cross-validation accuracy? Why? Please give specific arguments.

(f) Using Logistic Regression Model to perform 3 fold cross-validation on the training set and report your average classification accuracy.

(g) Train your best-performed classifier with all of the training data, and generate test labels on test set. Submit your results to Kaggle and report the accuracy.

WRITTEN EXERCISES

You can find links to the textbooks for our class on the course website.
Submit the answers to these questions along with your writeup as a single `.pdf` file. We do recommend you to type your solutions using LaTeX or other text editors, since hand-written solutions are often hard to read. If you handwrite them, please be legible!

1. **HTF Exercise 4.1** (From Generalized to Standard Eigenvalue Problem)

2. **HTF Exercise 4.2** (Correspondence between LDA and classification by linear least squares.)

3. **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.