Homework 1

Machine Learning for Signal Processing - JHU

Due: October 9th 23:59

Problem 1: Face Detection

This problem has three parts.

Part 1: Simple Face Detector

I. **Goal :** You will implement a simple face detector that can detect faces in group photos of people.

II. **Data :**

- Training Dataset: You will use the LFWCrop database as a training dataset. You have been provided a training dataset of face images from the LFWCrop Database in the Homework Template (`problem_1/data/lfw1000`).
- Testing Dataset: You have been provided testing data in the form of four photos of people in a group (`problem_1/data/group_photos`).

III. **Methodology:**

1. Compute the first Eigenface from training data: To do so, read all the images into a matrix, and then compute the first Eigenvector for this matrix.
2. To detect faces in image: To do so, you must scan the group photos and identify all regions that match the pattern of the first Eigenface the most. To **scan** the group photo to **match** against an 64×64 Eigenface, every 64×64 region of the photo must be compared against the Eigenface. The test images are much bigger than 64×64.
3. The **match score** between any 64×64 region of the group photo and the Eigenface is given by the **normalized dot product** between them. The dot product of the unrolled vector form of the Eigenface E and the unrolled 64×64 image patch P is given by $\frac{E.P}{|P|}$.
4. The locations of faces are likely where the **match score** has peaks across the group photos.
5. **Scaling:** The Eigenface you find is fixed in size and can only be used to detect faces of the same size as the Eigenface itself. In the group photos, faces are of different sizes. To solve this, scale the group images into many different sizes and scan them all with the Eigenface.
6. Resize each group photo by a factor of 0.5, 0.75, 1.5 and 2.0. Therefore, there are a total of 5 versions of each group photo. Scan all versions with the Eigenface, and calculate **match scores** for all of them, and locate peaks in all of them.
7. Sometimes, multiple peaks will occur in the same location (or within a few pixels of each other), in which case you can merge them.
8. Plot all the detected faces in the original group photo images.

IV. **Points to Note:**

- Some of the group photos are in color, whereas the Eigenface is greyscale. You will have to convert the color photographs to greyscale by setting the grey value as the mean of the red, green and blue values.
- For peaks that have been detected in the scaled group photos, you can find their corresponding positions in the original size image by de-scaling. If you found a peak at (X, Y) in the 2x image, the pixel positions in the original image are at $(X/2, Y/2)$. The size of the face in the original image will be $N/2 \times M/2$.
- Additional heuristics can also be implemented for comparison of peak values from different scale factors, appropriate thresholding for peak detection, additional scaling factors, etc. These are for you to experiment with.

Part 2: Boosting-based Face Detector

I. **Goal :** You will implement an Adaboost Classifier to classify between face images and non-face images.

II. **Data :**

- Training Dataset: You are given a training dataset of images (`problem_1/data/boosting_data/train`). You will also be re-using the LFWCrop dataset from Part 1 to generate Eigenfaces (more than the 1st one this time) (`problem_1/data/lfw1000`).
- Testing Dataset: You are given a testing dataset of images (`problem_1/data/boosting_data/test`).

III. **Methodology :**

1. Step one is to learn the first K Eigenfaces from the LFWCrop Dataset. Set $K = 10$ initially, but vary it appropriately to get the best results. This time, we will normalize the images to zero mean and unit variance before computing Eigenfaces.
2. Represent all the face images in `/train/face` as a linear combination of the Eigenfaces calculated above. Therefore, write each face F_i as

$$F_i \simeq w_{i,1}E_1 + w_{i,2}E_2 + ... + w_{i,K}E_K \quad (1)$$

   where E_j is the jth Eigenface and $w_{i,j}$ is the weight of the jth Eigenface in composing the ith face image. (Note that $w_{i,j}$ is simply the dot-product between F_i and E_j).
3. Represent each face F_i in your training set by the weights for the Eigenfaces, therefore

$$F_i \rightarrow \{w_{i,1}, w_{i,2}, w_{i,3}, ..., w_{i,K}\} \quad (2)$$

4. Similarly, for all the non-face images in your training data present in `/train/non-face`, represent them as linear combinations of the Eigenfaces as well, and use the weights of these Eigenfaces to represent these images. Therefore, for a non-face image NF_i, we have

$$NF_i \simeq v_{i,1}E_1 + ... + v_{i,K}E_K \quad (3)$$

   as before, the weights $v_{i,j}$ can be computed as dot-products, and the non-face images can be represented as

$$NF_i \rightarrow \{v_{i,1}, v_{i,2}, ..., v_{i,K}\} \quad (4)$$

5. The set of weights for the Eigenfaces for all images are the features representing these images. Using these weights as features, learn an Adaboost classifier to classify between face and non-face images.
6. Report your overall classification accuracy on the combined testing data present in `/test` by classifying these images using your Adaboost classifier, and comparing with the true labels.

Part 3: Gender Detection

I. **Goal:** You will build a gender detection system using the PCA dimensions from images.

II. **Data:**

- Training Dataset: You have been provided a training dataset of images split into male and female faces in `problem_1/data/lfw_genders/male/train` and `problem_1/data/lfw_genders/female/train`.
- Testing Dataset: You have been provided a testing dataset of images split into male and female faces in `problem_1/data/lfw_genders/male/test` and `problem_1/data/lfw_genders/female/test`.

III. **Methodology:**

1. Load the training data in `/male/train` and `female/train`.
2. Change each image into a vector by reshaping it, and combine all images into an image matrix.
3. Find the k best PCA dimensions of this data by performing PCA analysis on the image matrix (calculate the eigenvectors and eigenvalues for the correlation matrix of the image matrix). Here $k \in \{50, 100, 200, 300\}$.
4. Plot all the eigenvalues of the correlation matrix for the image matrix, and suggest which value of k you think captures the most information from the training dataset.
5. Calculate and plot the average male face and the average female face by finding the mean of all the male training images and all the female training images.
6. Develop a simple gender detection based on these average faces as follows
   i. Project your average male and female face on your k PCA dimensions.
   ii. Project all your testing images on your k PCA dimensions.
   iii. Classify each testing image as male or female by finding the distance between the weights of each testing image and the weights of your average male face and average female face.
   iv. Calculate the accuracy of classification across your testing data for different values of $k \in \{50, 100, 200, 300\}$ used for projection. Briefly explain the time vs. accuracy trade-off for different values of k that you see.
   v. **Note:** You *can* compare the projected faces of each testing face to the projected average male and female faces as well. However, comparing weights is considerably faster, and these are equivalent if your eigenvectors from PCA are orthonormal. MATLAB's `eig()` function returns normalized eigenvectors, but `svd()` *does not!* You will have to manually normalize the eigenvectors if you use the `svd()` function.
7. Develop a gender detection algorithm based on *all* training images, as follows -
   i. Project **all** the training images into the k PCA dimensions, and *all* the testing images into the k PCA dimensions as well.
   ii. Calculate the average Euclidean distance between the weights of each test image and weights of **all** training male images and **all** train female images. Classify the test image based on which average Euclidean distance is smaller.
   iii. Same as before, calculate the accuracies for $k \in \{50, 100, 200\}$. Argue about the time vs. accuracy trade-off, which will be a lot more pronounced in this case.
   iv. **Note:** You *will* have to optimize your Euclidean distance calculation, or else your code will take too long to run. Matrix operations in MATLAB are your friend. Same

as above, you *can* compare projected images directly, but comparing weights is much faster, as long as eigenvectors are normalized.

Problem 2: Music Transciption and Projection

You are required to transcribe the music from the song `problem_2/data/Polyushka.wav` making use of a set of notes played with harmonica, included in the folder `problem_2/data/notes_15`. You must determine how each of the notes is played within the song. All the files required for this exercise can be found in the homework template. The recordings have been downloaded with permission from the artist from YouTube. The `helper_code.pdf` file inside the main folder in the homework template contains information about how to convert each note into a spectral vector, and the entire music to a spectrogram matrix.

Part 1. Analysis by individual note:

1. Compute the contribution of each individual note to the entire music. Consider that if N_i is the vector representing the i^{th} note and $\mathbf{M}$ is the music matrix, find the row vector (transcription) W_i such that $N_i W_i \approx \mathbf{M}$. Return the transcription for each separated note. Here, you're trying to approximate $\mathbf{M}$ by just one note N_i.
2. Recompose the music employing the transcription you just estimated for all the notes individually. Mathematically, compute $\hat{M} = \sum_i N_i W_i$ (add up the individual reconstructions to get the final reconstruction) and invert the result to produce music. To invert it to a music signal, follow the instructions given in `helper_code.pdf`. Return the recomposed signal in a `.wav` file. Comment about how the recomposed music compares to the original signal. How could the result be improved?

Part 2. Joint analysis using all notes:

1. Now, and using a different technique, compute the contribution of all notes to the entire music jointly. Mathematically, if $\mathbf{N} = [N_1, N_2, ...]$ is the notes matrix where the individual columns are the notes, find the matrix $\mathbf{W}$ such that $\mathbf{NW} \approx \mathbf{M}$. The ith row of $\mathbf{W}$is the transcription of the ith note.
2. Recompose the music employing the transcription you just estimated and the notes. Mathematically, compute $\hat{M} = \mathbf{NW}$, and invert the result to produce music. To invert it to a music signal, follow the instructions given in `helper_code.pdf`. Return the recomposed signal in a `.wav` file. Is the recomposed music identical to the music you constructed in Part 1.2? Explain your conclusions.

Problem 3: Optimization and non-negative decomposition

Non-negativity is an inherent property of certain types of signals e.g. muscle activity or audio spectrograms. In this problem we would be performing a non-negative decomposition of a music spectrogram. But first, let's begin with some basic understanding of the theory and build a motivation.

A music spectogram is a visual representation of the frequency of sounds that make up a certain duration of music. Spectograms are simply Fourier Transforms (to get the frequency components) at each time stamp stacked together, defined by the length of the audio. Fig. 1 shows a standard spectogram of a male voice. Music ($\mathbf{M}$) can be said to have been composed of a linear combination of notes $\mathbf{N}$ and their corresponding weights $\mathbf{W}$ and can be represented as:

$$\mathbf{M} = \mathbf{NW} \tag{5}$$

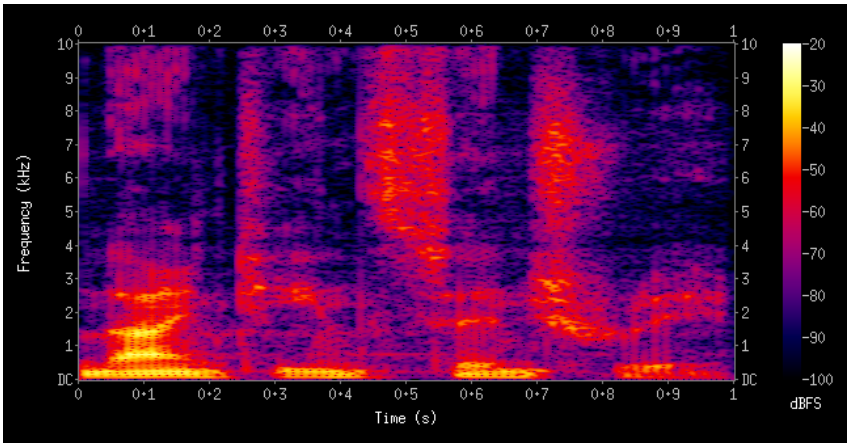

Figure 1: Spectrogram of a male voice saying the words — nineteenth century. Frequency is shown in an increasing order along the Y axis and time is on the X axis.

This is similar to the PCA lab, where a linear combination of the eigenvectors or eigenfaces are used to reconstruct a face. Or in case of the Polyusha Polye music, the notes can be the set of notes from a harmonica. Till now, fair and good. Now let's jump to the main issue.
Consider this — assume a piece of music $\mathbf{M}$ that has been played on an acoustic guitar and we have all the notes $\mathbf{N}$ that an acoustic guitar can play, we can get an approximate of the original music by representing the reconstructed music $\hat{M}$ as:

$$\hat{M} \approx \mathbf{NW} \tag{6}$$

where the goal would be to find the weights ($\mathbf{W}$) which defines the amount each note $\mathbf{N}$ contributes to a snippet of audio. To obtain these weights ($\mathbf{W}$), we perform the simple linear algebra as follows:

$$\mathbf{W} \ = pinv(\mathbf{N})\mathbf{M} \tag{7}$$

Note that the above equation is derived from the notes ($\mathbf{N}$) and music ($\mathbf{M}$) already known to us and comes from Eqn. 5. At time-step i, the i^{th} column of $\mathbf{M}$ can be approximated as:

$$M_i = \Sigma_j N_j W_{j,i} \tag{8}$$

where N_j, is the j^{th} note in $\mathbf{N}$ and $W_{j,i}$ is the weight assigned to the j^{th} note in composing the i^{th} frame of music.
The issue often is that while computing $\mathbf{W}$ (in eqn. 7), we might have negative values which intuitively does not make sense as notes are are additive and cannot be *unplayed* as such to create music. This also results in the approximated music $\hat{M}$ (in eqn. 6) to have some negative values. Spectral magnitudes cannot be negative!
So, now that we are aware of the issue, let's try and find a way out of it. The most logical way to approach this problem is to put a non-negative constraint on the weights $\mathbf{W}$, such $W_{i,j} \geq 0$.
Solving the problem for an exact solution, with the constraint, is non-trivial and therefore numerical approximations are used to find the weights $\mathbf{W}$. The most common method is by gradient descent, where we minimize $||\mathbf{M\text{-}NW}||_F^2$ with respect to the constraint that W is non-negative. Note $||A||_F$ refers to the Frobenius Norm. Let's dive into the algorithm:

(a) ***Derivative computation*** - The error function between the original music $\mathbf{M}$ and the approximated $\hat{M}$ can be given as:

$$E = \frac{1}{FT}||\mathbf{M} - \mathbf{NW}||_F^2 \tag{9}$$

where F is the dimensionality of $\mathbf{M}$, and T is the number of frames/time-steps.
Differentiate the above equation of error (E) for this part w.r.t $\mathbf{W}$ — $\frac{dE}{d\mathbf{W}}$. Derive the differential and show the steps involved.

(b) ***Gradient Descent*** - Let $\mathbf{W}^0$ be the initial estimate of $\mathbf{W}$ and $\mathbf{W}^n$ the estimate after $\boldsymbol{n}$ iterations. Use the following update rule to iteratively change the weight as:

$$\hat{\mathbf{W}}^{n+1} = \mathbf{W}^n - \eta \frac{dE}{d\mathbf{W}}|_{\mathbf{W}^n} \tag{10}$$

where $\frac{dE}{d\mathbf{W}}|_{\mathbf{W}^n}$ is derivative of E w.r.t. $\mathbf{W}$ when $\mathbf{W}$=$\mathbf{W}^n$. To ensure the non-negative nature of $\mathbf{W}$ we define add the additional constraint as follows

$$\mathbf{W}^{n+1} = max(\hat{\mathbf{W}}^{n+1}, 0) \tag{11}$$

where $max(\hat{\mathbf{W}}^{n+1}, 0)$ is the component-wise flooring operation that sets all negative entries to 0.

Implement the above algorithm by initializing $\mathbf{W}$ as a unity matrix, η as (0.0001 and 0.01). Run for 500 iterations in each case. Plot 1) E as function of iteration number n and 2) E as a function of η. Your code should return these plots and the final weights $\mathbf{W}$.

(c) ***Music reconstruction*** (optional)- From the results obtained in the previous step, use the best η (one with the lowest error) to recreate the music from the equation given in eqn. 6. What difference do you hear with the approximated music? Write your comments within the .m file in no more than 2 sentences.

Submission Instructions

Please follow these instructions **closely**. We **will** deduct points for not sticking to the template. Solutions should be uploaded to Blackboard. `HW1_template.zip` contains all the data and necessary templates required to solve all the problems.

Problem 1: Face Detection

The `problem_1` directory has a a `data/` folder, and three sub-folders, one for each part.

(a) **Part 1:** `problem_1/part_1/` has a driver file called `run_part_1.m`. We should be able to execute this file to output **all** the required plots and images from Part 1, assuming the directory structure as we've specified.
Note: `problem_1/part_1/helper_code.pdf` contains some useful tips and tricks to help you out in Part 1.

(b) **Part 2:** `problem_1/part_2/` has a driver file called `run_part_2.m`. We should be able to execute this file to output **all** the required plots and images from Part 2, assuming the directory structure as we've specified.

(c) **Part 3:** `problem_1/part_3/` has a driver file called `run_part_3.m`. We should be able to execute this file to output **all** the required plots and images from Part 3, assuming the directory structure as we've specified.

(d) **Problem 1 Writeup:** Submit a file `problem_1_writeup.pdf` containing **all** the images you are required to plot, and all the written questions and points you're supposed to discuss.

Problem 2: Projections

The `problem_2` directory has a `data/` folder, and one sub-folder `problem_2/results`, to include the results. The main directory also has a template file called `problem_2/run_problem_2.m`. Write your code into this script. There are instructions within the script on how to save your outputs. The scripts must be written so that we can simply run `run_problem_2.m` from MATLAB within the directory, without any additional commands or adding any additional file to the directories. If we cannot run the program, we cannot score you. **Note:** All yopur results should go in the `problem_2/results/` folder. Instructions for each part:

(a) **Part 1: Analysis by individual note**

1. Your solution will give you one score line or transcription per note. Place all the scores in a single matrix and save this as a single file named `problem2a.dat`

2. For this part, the synthesized music must be saved as `problem2b_synthesis.wav` within the results directory.

(b) **Part 2: Joint analysis using all notes**

1. Your solution will give you single score matrix. Save this as a single file named `problem2a.dat`

2. For this part, the synthesized music must be saved as `problem2b_synthesis.wav` within the `results` directory.

Problem 3:

We shall use the same data (`Polyushka.wav`) music used in the previous question. There is a "`problem_3/run_problem_3.m`" file that should output *only* the following:

(a) Plot of error as a function of iteration number for $\eta = 0.0001, 0.001, 0.01, 0.05$.

(b) Final weight matrix after 500 iterations

(c) (Optional) Approximated music saved as `polyushka_approx.wav`

(d) All outputs should be saved in the `problem_3/results` folder. Save the plots as PNG named as `error_eta_#.png` where `#` should correspond to the `eta` value used and the final weight matrix as `final_weight.dat`

(e) The derivation for $\frac{dE}{d\mathbf{W}}$ should be saved in `problem_3_writeup.pdf` file in the `results` folder.

Note: Ensure graphs have a title and are labelled properly. Points will be deducted if your code doesn't match the required output format mentioned above. Your code should not output anything else.