Homework 2

Data-driven Representations - Boosting - Face Detection

CMU 11-755/18-797: Machine Learning for Signal Processing (Fall 2022)

OUT: September 23rd, 2022
DUE: **October 13th, 11:59 PM EST**

START HERE: Instructions

- **Collaboration policy:** Collaboration on solving the homework is allowed, after you have thought about the problems on your own. It is also OK to get clarification (but not solutions) from books or online resources, again after you have thought about the problems on your own. There are two requirements: first, cite your collaborators fully and completely (e.g., “Jane explained to me what is asked in Question 3.4”). Second, write your solution <u>independently</u>: close the book and all of your notes, and send collaborators out of the room, so that the solution comes from you only.

- **Submitting your work:** Assignments should be submitted to Canvas unless explicitly stated otherwise. Please submit all derivation/explanation results as **report_{YourAndrewID}.pdf**. **Each derivation/proof should be completed on a separate page**. Submissions can be handwritten, but should be labeled and clearly legible. Alternatively, submissions are strongly encouraged to be typeset using LaTeX. Please refer to Piazza for detailed instructions for joining Canvas and submitting your homework.

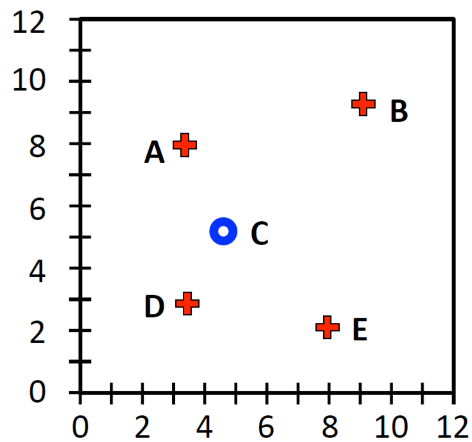
- **Programming**: All programming portions of the assignments should be submitted to Canvas as well. Please submit all codes and output files as **programming_{YourAndrewID}.zip**. We will not be using this for autograding, but rather for plagiarism detection, meaning you may use any language you would like to program.

- **Late submissions**: You have **in total 7 slack days** that you can **freely apply** to any homework. Any homework submitted after running out of slack days will receive zero credit. Please make sure you submit on time.

1 Understanding Boosting and ICA

1.1 Adaboost

The figure below shows a data set that contains two classes ('+' and 'o'). The instances are labeled A-E.



If we train Adaboost to solve the classification problem using decision stumps on each feature:

1. Which instances will have their weights increased at the end of the first boosting iteration? (Explain)
2. What is the **minimum** number of iterations that the algorithm could take to achieve zero training error? (Explain)

1.2 Affine Transformation of Random variables

Let $\mathbf{x}$ be a d-dimensional random vector with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. Let $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$, where $\mathbf{A}$ is an $n \times d$ matrix and $\mathbf{b}$ is a n-dimensional vector.

1. Show that the mean of $\mathbf{y}$ is $\mathbf{A}\boldsymbol{\mu} + \mathbf{b}$
2. Show that the covariance matrix of $\mathbf{y}$ is $\mathbf{A}\boldsymbol{\Sigma}\mathbf{A}^\top$

1.3 Difference between Correlation and Independence

Consider the discrete random variable X described as follows

$$p(X = x) = \begin{cases} 1/3 & \text{if } x = -1 \\ 1/3 & \text{if } x = 0 \\ 1/3 & \text{if } x = 1 \end{cases}$$

We also define the random variable $Y = 1 - X^2$.

1. Compute the values of $\mathbb{E}(X)$ and $\mathbb{E}(Y)$.
2. Compute $\mathbb{E}(XY)$ and $\text{Cov}(X, Y)$. Are X and Y uncorrelated?
3. Are X and Y independent? i.e., $p(X = x, Y = y) = p(X = x)p(Y = y)$ for all x, y?

2 Source Separation using Independent Component Analysis

In this problem, you will implement your own version of ICA and apply it to source separation.

You are given 4 audio recordings of music in which the individual tracks have been mixed with 4 different balances, i.e., relative volumes. These are in `hw2_materials_f22/problem2`. The song used is "Easy Tiger" by Kangoro obtained from the 'Mixing Secrets' Free Multitrack Download Library.

All of these recordings were generated by mixing different audio signals. Your objective is to reconstruct the original sounds using ICA. Do the following steps:

1. Implement your own version of ICA based on FOBI, the method that we discussed in class. Write a function, `ica`, which receives as input a $M \times N$ matrix and outputs a $M \times N$ matrix where its rows are the extracted independent components. Submit your code.
2. From the data folder, read the files `mix1.wav`, `mix2.wav`, `mix3.wav`, `mix4.wav` as mono audio, i.e., single channel audio, and extract the audio signals `s1`, `s2`, `s3`, `s4`. Make sure the four signals have the same length. Stack these signals to generate a matrix $\mathbf{M}$ with 4 rows and as many columns as there are samples. Apply the function `ica` on the matrix $\mathbf{M}$ and save the components generated as `source1.wav`, `source2.wav`, `source3.wav`, `source4.wav`. Don't forget, ICA does not consider scale factors, so you may need to boost or decrease the resulting signal to ensure it is audible and does not clip when played on speakers. Submit files `source1.wav`, `source2.wav`, `source3.wav`, `source4.wav`.
3. If $\mathbf{H}$ is an $M \times N$ matrix where its rows correspond to the output of `ica`, then we can say that

$$\mathbf{M} = \mathbf{AH}$$

   In this case, $\mathbf{A}$ is the mixing matrix which produces our observation $\mathbf{H}$. Compute the 4×4 mixing matrix for this case. Submit $\mathbf{A}$ as `mixing_matrix.csv`.

3 Face Detection

3.1 Preprocessing for Eigenface Computation

In the directory `hw2_materials_f22/problem3` you can find a folder named `lfw1000`, which contains 1071 face images. Each of these images is a 64×64 gray scale image. Figure 1 shows some examples.

Figure 1: Examples of face images.

The Matlab (top) and Python (bottom) command to read an image is:

```
image = double(imread(imagefile));
----------------------------------------------------------------------------------------------------
from PIL import Image
image = Image.open(imagefile)
```

Note we are using `double()`. If you do not use it, Matlab reads the data as `uint8` and some operations cannot be performed.

We consider that each face can be approximated by a linear combination of eigenfaces, that is, each face F can be approximated as

$$F \approx \omega_{F,1}E_1 + \omega_{F,2}E_2 + ... + \omega_{F,k}E_k \tag{1}$$

where E_i is the i^{th} eigenface and $\omega_{F,i}$ is the weight of the i^{th} eigenface when composing face F.

To learn eigenvectors, the collection of faces must be unravelled into a matrix. To unravel an image of any size, you can do the following:

```
[nrows, ncolumns] = size(image);
image = image(:);
----------------------------------------------------------------------------------------------------
import numpy as np
nrows, ncolumns = image.height, image.width
image = np.asarray(image.getdata())
```

The first line above is used only to retain the size of the original image. We will need it to fold an unravelled image back into a rectangular image. The second line converts the `nrows` $\times$ `ncolumns` image into a single `nrows`$\cdot$`ncolums` $\times$ 1 vector. To read in an entire collection of images, you can do the following:

```
filenames = textread('FILE WITH LIST OF IMAGE FILE NAMES','%s');
nimages = length(filenames);
for i = 1:nimages
    image{i} = double(imread(filenames{i}));
end
----------------------------------------------------------------------------------------------------
# Feel free to use pathlib library instead of the os library if you like pathlib better.
# pathlib is more modern.
import os
```

```
for i, filename in enumerate(os.listdir(imagedirectory)):
    image = Image.open(os.path.join(imagedirectory, imagefile))
    X[:,i] = np.asarray(image.getdata())
```

To compose matrix from a collection of `k` images, the following Matlab script can be employed (you can also do it your own way, not necessary if using the Python script above):

```
X = [];
for i = 1:k
    X = [X image{i}(:)];
end
```

Eigenfaces can be computed from `X`, which is an (`nrows` · `ncolumns`) × `nimages` matrix. You will extract eigenfaces using three different basis representations in the following sections. Nothing needs to be submitted for this section.

3.2 Computing Eigenfaces with Karhunen-Loève Expansion

For this part of the assignment, you will extract eigenfaces using KLE, which employs singular value decomposition to project the faces onto orthogonal bases (in Matlab, you can use the command `svd`; in Python, `np.linalg.svd`).

Each resulting eigenface will be in the form of a (`nrows` · `ncolumns`) × 1 vector. To convert it to an image, you must fold it into a rectangle of the right size. Matlab will do it for you with:

```
eigenfaceimage = reshape(eigenfacevector, nrows, ncolumns);
-------------------------------------------------------------------------------------------------
eigenfaceimage = eigenfacevector.reshape(nrows, ncolumns)
```

`nrows` and `ncolumns` are the values obtained when you read the image. `eigenfacevector` is the eigenvector obtained from the Eigenanalysis (or SVD).

1. **First Eigenface.**

   Using these 1071 images, find the first eigenface and plot it in your report (you can use the Matlab command `imagesc` or the Pyplot command `plt.imshow()`). Submit the first eigenface as a 4096×1 vector in a file named `eigenface.csv`.

2. **Reconstruction Error.**

   Given a face F and the first k eigenfaces, we can compute its reconstruction error as

   $$\mathcal{E}(F\ ;\ E_1, ..., E_k) = \left\| F - \sum_{i=1}^{k} \omega_{F,i} E_i \right\|_2 \tag{2}$$

   Then, if we have N faces, the mean reconstruction error is given by

   $$\frac{1}{N} \sum_{j=1}^{N} \mathcal{E}(F_j\ ;\ E_1, ..., E_k) \tag{3}$$

   Using the 1071 provided images, plot the mean reconstruction error as function of k. Consider k from 1 to 100. Attach this plot to your report. Report the mean reconstruction error using $k = 100$.

3.3 Computing ICA Faces with FOBI

Another set of bases for images can be determined by using ICA, which extracts statistically independent bases from the images. In problem 2, you used the FOBI implementation of ICA to perform blind source separation on a mixed signal. Now you will explore the feature extraction capabilities of ICA by calculating ICA faces using FOBI.

Before you get started, you should be aware of some implementation details and hints. First, there are different ways of viewing the images of the given face dataset. In one view, all the pixels in one face are viewed as a single sample; in the other view, all the pixels at a given coordinate position across all images are considered to be a single sample. Both of these views are useful in extracting ICA faces, but the results will be fundamentally different. We recommend that you experiment with both of these views, but, for consistency, **we ask that you take the first view, where each image is a sample, for this homework.**

Second, computational implementations of eigendecomposition can sometimes result in poor approximations which lead to incorrect ICA bases. If you believe you are experiencing issues because of eigendecomposition, consider how SVD might serve as a substitute.

Third, remember that ICA acts on centered data, not the original data. Keep this in mind as you project images onto your ICA bases.

Lastly, note that extracting too many independent components from the image dataset can yield poor results. You can account for this in your FOBI algorithm by considering only the first n eigenvalues of the correlation matrix. **For this homework, you should extract n = 100 ICA faces.**

1. **ICA Face Visualization.**

   Using the 1071 images, calculate the ICA faces using FOBI, making changes to your implementation from problem 2 as needed. Plot the first ICA face from your results and put it into your report.

2. **Reconstruction Error.**

   As you did for KLE, plot the mean reconstruction error from the ICA representation as a function of k, considering k from 1 to 100. Attach this plot to your report. Report the mean reconstruction error using $k = 100$.

3. **Comparing KLE and ICA.**

   ICA, of course, is fundamentally different from KLE. To show this difference mathematically, one could simply calculate the angle between two basis vectors to check if the bases are constrained to be orthogonal or not. Calculate the angle between your first two eigenfaces, then do the same for the first two ICA faces. Include these numbers in your report.

   Next, another fundamental difference between KLE and ICA can be manifested in how the reconstruction error decreases with the number of KLE or ICA components incorporated, respectively. Include in your report a few sentences explaining why the reconstruction error curves look the way they do for KLE and ICA.

3.4 Adaboost Implementation

In class we saw that Adaboost allows us to train a strong classifier combining weak classifiers, taking the following form:

$$F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t \, h_t(\mathbf{x}) \tag{4}$$

$$H(\mathbf{x}) = \operatorname{sign}(F(\mathbf{x})) \tag{5}$$

where h_t is the t-th weak classifier and α_t the weight computed in the training phase. In this part, you will implement your own version of Adaboost.

1. Write the function `adaboost_train` which receives as inputs:
   - `X_tr`: a $N \times M$ matrix, where each row corresponds to a data sample with M dimensions. This is the training data.
   - `Y_tr`: a $N \times 1$ matrix, which contains 1s and -1s only. This vector defines the class label. The i component is the class corresponding to the i-th row of `X_tr`.
   - `T`: an integer number which defines the number of weak classifiers to use.

   The output of this function must be `model`. This can be any data structure that contains all the information you need to make the inference. If you plan to work with Matlab, we recommend to use structures.

2. Write the function `adaboost_predict` which receives as inputs:
   - `model`: this is the output obtained from function `adaboost_train`
   - `X_te`: a $P \times M$ matrix, where each row corresponds to a data sample with M dimensions.

   The output of this function must be:
   - `pred`: a $P \times 1$ vector, where each component is a real number. The i-th component is the prediction score F (equation 4) for the i-th row of `X_te`. Note that the actual prediction will be `sign(pred)`.

   Submit your code.

3.5 Training Adaboost

In this part, you get to use your implementation of Adaboost.

In the directory `hw2materials_f22/problem3` you can also find 2 folders: `train` and `test`. In each of these you can find two folders: `face` and `non-face`. In the folder `face` there are pictures of faces, while in the folder `non-face` there are non faces images. Each of these images is a 19×19 gray scale image.

We use the eigenfaces to represent each image as a real value vector. As discussed in class, we can project the image I on the eigenface E_i and use the corresponding weight $\omega_{I,i}$ as one component of this representation. Formally, an image I is represented as follows

$$\mathcal{R}(I\,;\,E_1, ..., E_k) \quad = \quad \begin{pmatrix} \omega_{I,1} \\ \omega_{I,2} \\ \vdots \\ \omega_{I,k} \end{pmatrix} \tag{6}$$

To do this, the image and the eigenfaces must have the same size.

1. Rescale the images from folder `lfw1000` to a 19×19 size. You can use the command

   ```
   image = imresize(image,[19,19]);
   ------------------------------------------------------------------------------------
   image = image.resize((19,19))
   ```

   Then, compute new 19×19 eigenfaces and ICA faces using this new set.

2. Considering only the first $k = 10$ eigenfaces, calculate the projection weights by projecting every image onto each eigenface basis, and use these values as features for your classifier. With this configuration, each sample image should result in a feature vector with 10 components. To define `Y_tr`, represent each face image with 1 and each non-face image with -1. Train your model using the representation of the images in the `train` folder with different values of T, the number of weak classifiers, in the list (10, 50, 100, 150, 200). Plot your classification errors both in the training and testing set as a function of T. Attach this plot to your report.

3. Repeat the previous step for $k = 30$ and $k = 50$. Is there any improvement? Attach the corresponding plots to your report.
4. Now, repeat the previous experiment for the same values of k in (10, 30, 50) and T in (10, 50, 100, 150, 200), but swap out the eigenface representation for the ICA representation. How do the two representations compare? Attach your comments on this question and the new classification error plots to your report.

4 Music Decomposition using Principle Component Analysis

Employing the same spectrogram computed for `FoxTitle.wav`, which is 61-frame by using a window size of 1024 samples and a hop size of 160 samples in STFT. This time we are going to perform PCA to attempt music transcription, then take a look at how good a job PCA can do. `FoxTitle.wav` and files of the ground truth notes are under `hw2materials_f22/problem4`.

Let $\mathbf{M}$ be the (magnitude) spectrogram of the music, which is of size $D \times T$, where D is the FFT size and T is the total number of windows in Fourier analysis, which represents the length of music, the sample mean $\boldsymbol{\mu}$ be the mean spectrum among all Fourier analysis windows, which is of size $D \times 1$, the sample covariance $\mathbf{S}$ is thus:

$$\mathbf{S} = \frac{1}{T}(\mathbf{M} - \boldsymbol{\mu})(\mathbf{M} - \boldsymbol{\mu})^\top$$

We then perform eigendecomposition on the sample covariance matrix $\mathbf{S}$:

$$\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}$$

Here the eigenvector matrix $\mathbf{U}$, which is orthonormal and consisting of $\mathbf{u}_1, \mathbf{u}_2, \dots \mathbf{u}_D$ as eigenvectors, and the eigenvalue matrix $\boldsymbol{\Lambda}$, on whose diagonal all eigenvalues $\lambda_1, \lambda_2, \dots \lambda_D$ are highly likely positive as $\mathbf{S}$ is highly likely positive-definite, are both of size $D \times D$. Based on these orthonormal eigenvectors, $\mathbf{M}$ can be constructed in the way of using the notes and weights:

$$\mathbf{M} = \mathbf{U}\mathbf{Z} + \boldsymbol{\mu}$$

Where $\mathbf{Z}$ is of size $D \times T$ and consisting of $\mathbf{z}_1^\top, \mathbf{z}_2^\top, \dots \mathbf{z}_D^\top$ as row vectors. Each $\mathbf{z}_i^\top$ contains the projection lengths of all frame instances, i.e., column vectors in the centered spectrogram $\mathbf{M} - \boldsymbol{\mu}$, onto each $\mathbf{u}_i$.

The punchline is that the eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \dots \mathbf{u}_D$ serve as the principle components for the original data, which is the spectrogram $\mathbf{M}$ in this scenario. Each eigenvalue λ_i is the variance of the projection lengths in $\mathbf{z}_i^T$, as all frame instances of the centered spectrogram $\mathbf{M} - \boldsymbol{\mu}$ are projecting onto each principle direction defined by $\mathbf{u}_i$. The larger the variance λ_i is, the more significant this principle component $\mathbf{u}_i$ amounts to be.

Ideally, if each base vector in $\mathbf{U}$ can represent one pitch used to compose the music, it would be easy to tell how many pitches there are by the magnitudes of eigenvalues, as major principle components have significant eigenvalues and carry more information.

1. Use STFT to analyze the music signal. Please use a window size of 1024 samples and a hop size of 160 samples. You can analyze the signal either in linear scale or in logarithmic scale, and with any other STFT parameters you find that perform the best. Then, decompose the signal with PCA and fetch the 11 most significant bases in $\mathbf{U}$.

   Submit the 11 most significant components of matrix $\mathbf{U}$ as `musicbases.csv`.

2. In the folder `hw2materials_f22/problem4` we also have the ground truths of the individual notes for you to validate your results. Using a windows size of 1024 samples and a hop size of 160 samples, every individual note recording will be segmented to 62 frames. Here we view the part starting from the 21st frame to the 40th frame as the steady part with a consistent spectrum. The average spectrum of these frames is the spectrum of this building block note. Compare the 11 building blocks you've got using PCA with each of the ground truth notes by calculating the absolute value of the cosine similarity between your bases and each of the ground truth bases to find the match with the highest similarity score. How many unique notes can you actually restore and how close are your answers? Please report the note names of the best match of your 11 bases, and their respective absolute cosine similarity scores. Write your thoughts about why PCA is / isn't proper for this task.