Homework 2
Data-driven Representations - Boosting - ICA

CMU 11-755/18-797: Machine Learning for Signal Processing (Fall 2019)

OUT: September 24th, 2019
DUE: **October 8th, 11:59 PM**

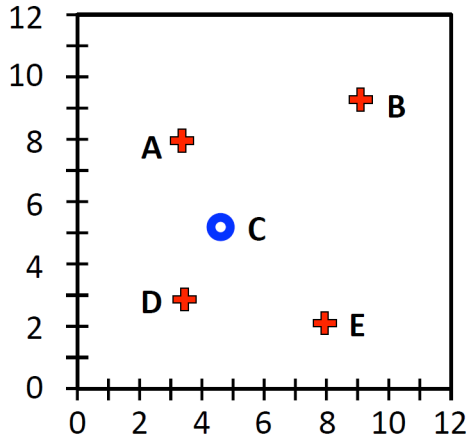
START HERE: Instructions

- **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 independently: 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 as PDFs using Canvas unless explicitly stated otherwise. Please submit all results as "report_YourAndrewID.pdf" in you submission. **Each derivation/proof should be completed on a separate page**. Submissions can be handwritten, but should be labeled and clearly legible. Else, submissions can be written in LaTeX. Upon submission, label each question using the template provided by Canvas. Please refer to Piazza for detailed instruction for joining Canvas and submitting your homework.
- **Programming**: All programming portions of the assignments should be submitted to Canvas as well. Please `zip` or `tar` all the code and output files together, and submit the compressed file together with the pdf report. We will not be using this for autograding, meaning you may use any language which you like to submit.
- **Late submissions**: Any homework submitted after the deadline will receive no credit. Please make sure you submit on time.

1 Understanding Adaboost 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.

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)
3. Is it possible to add another example to the data set to guarantee that boosting achieves zero training error in just two iteration? If so, how? If not, why?

1.2 Affine Transformation of Random variables

Let $\mathbf{X}$ be a d-dimensional random vector with mean μ and covariance matrix Σ. Let $\mathbf{Y} = A\mathbf{X} + \mathbf{b}$, where A is a $n \times d$ matrix and $\mathbf{b}$ is a n-dimensional vector.

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

1.3 Difference between Correlation and Independence

Consider the discrete random variable X described as follows

$$\mathbb{P}(X = i) = \begin{cases} 1/3 & \text{if } i = -1 \\ 1/3 & \text{if } i = 0 \\ 1/3 & \text{if } i = 1 \end{cases}$$

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 $Cov(X, Y)$. Are X and Y uncorrelated?
3. Are X and Y independent? i.e., $\mathbb{P}(X = i\,,\, Y = j) = \mathbb{P}(X = i)\mathbb{P}(Y = j)$?

2 Eigen Faces and Face Detection

2.1 Compute EigenFaces

In the directory `hw2material/problem2` you can find a folder named `lfw1000`, which contains 1071 face images. Each of these images is a 64×64 gray scale image. The figure 1 shows some examples.

Figure 1: Examples of face images.

The matlab command to read an image is:

```
image = double(imread(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, this 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} Eigen face and $\omega_{F,i}$ is the weight of the i^{th} Eigen face, when composing face F.

To learn eigen vectors, 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(:);
```

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`·`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
```

To compose matrix from a collection of `k` images, the following matlab script can be employed (you can also do it your own way):

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

Eigen faces can be computed from `X`, which is an (`nrows` · `ncolumns`) × `nimages` matrix. There are two ways to compute eigen faces; one is using eigen analysis and the other is by singular value decomposition (in matlab, you can use the commands `eig` or `svd`).

The eigen face 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);
```

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

1. **First Eigen Face**.

   Using these 1071 images, find the first eigen face and plot it in your report (you can use the matlab commnad `imagesc`). Submit the first eigen face 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

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

   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 $k = 100$.

2.2 Adaboost Implementation

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

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

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

where h_t is the t-th weak classifier and α_t the weight computed in the training phase. In this part, you have to 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`: a integer number. It 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.

2.3 Training Adaboost

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

In the directory hw2materials/problem2/ 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 we discuss 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, we need that the image and the eigen faces 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]);
   ```

   Then, compute new 19×19 eigenfaces using this new set.

2. Considering the first $k = 10$ eigenfaces, for every image compute the projection to each eigen face, and use these values as features for your classifier. Thus, for each image you have to compute a vector with 10 components. To define `Y_tr`, consider 1 for face and -1 for non-face. Train your model using the representation of the images in the `train` folder and using different values of `T` (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 and report the previous step taking $k = 30$ and $k = 50$. Is there any improvement? Attach the corresponding plots to your report.

2.4 Face Detector

The task of face detector is finding the location of faces in a give image. As we learned from the class, a typical face detector has following steps.

1. Convert the given image to gray scale and rescale it to different sizes.

2. For each scale, use a face/non-face classifier to "scan" the images in a sliding window manner. Here the classifier is trained in section 2.3. Note that we use the predicted score (equation 4) instead of the predicted label of the AdaBoost classifier. With the response score for each window, we have a score map for each scale. Eventually we have a bunch of score maps in different scales. Each score in the map indicates the possibility that the patch in this window is a face. The windows are also called bounding box.

3. Find the local maximums in the score map and compare them to a threshold. Remove the ones with scores less than threshold.
4. Compute the corresponding bounding boxes based on the scores. Fuse the results from score maps of different scales.
5. Use non-maximum suppression (NMS) to find the potential bounding boxes. NMS is a post-processing algorithm that merges all bounding boxes that belong to the same face.

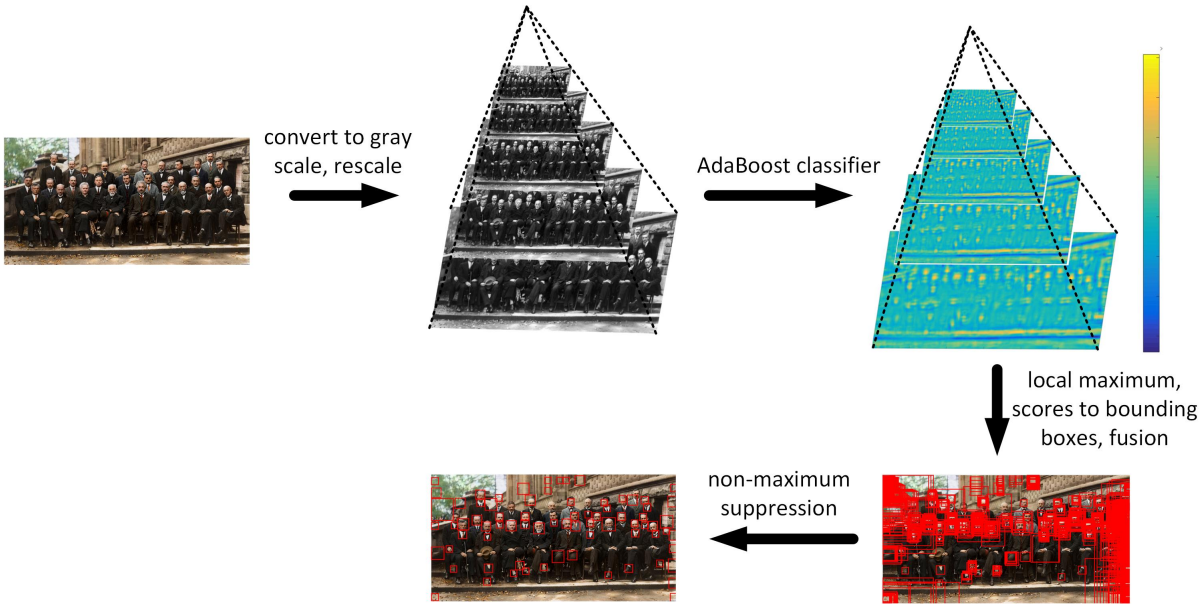


Figure 2: Pipeline for Face Detection.

It would be more intuitive in fig. 2. In this homework, step 3 and 5 is provided through two matlab functions that you can find in `h2materials/problem2/utils`:

- `get_localmax`: This function return the locations which may have a face. The candidate locations must be the local maximum, and their scores are greater than a pre-defined threshold.
- `nms`: This function implements non maximum suppression (NMS), a post-processing algorithm for merging the bounding boxes that belong to the same face. For each loop, it takes a bounding box with highest score, and removes the rest of bounding boxes that overlap with it.

Details about inputs and outputs can be found on the corresponding files.

You need to implement step 1, step 2, and step 4, and build the entire face detection system.

To test your system, 4 images are provided in the directory `hw2materials/problem2/photos`. Using these images, show your best results and the running times in the report. You are allowed to tune different parameters (number of eigenfaces, number of weak learners, threshold, etc). The result should be in the form of image with the predicted bounding boxes.

3 Source Separation using ICA

In this problem, you have to implement your own version of ICA and apply it to source separation.
You are given 2 audio recordings, `sample1.wav` and `sample2.wav`. These can be found in the directory `hw2materials/problem3.`

These recording were generated mixing two different audios. Your objective is to reconstruct the original sounds using ICA. Do the following steps:

1. Implement your own version of ICA based on FOBI (Freeing Fourth Moments) method that we discussed in class. Write a function `ica` which receives as input a $2 \times N$ matrix and outputs a $2 \times N$ matrix where its rows are the extracted independent components. Submit your code.

2. Read the file `sample1.wav` and extract the audio signal `s1`. This should be a vector with 132,203 components. Read the file `sample2.wav` obtaining the signal `s2`. Both `s1` and `s2` have the same size. Transpose and concatenate these signals generating a matrix $\mathbf{M}$ with 2 rows and 132,203 columns. Apply the function `ica` on the matrix $\mathbf{M}$ and using the Matlab function `audiowrite`, save the components generated as `source1.wav` and `source2.wav` respectively. Don't forget ICA does not consider scale factors, so you may need to boost or decrease the resulting signal. Submit files `source1.wav` and `source2.wav.`

3. If $\mathbf{H}$ is a $2 \times N$ where its rows correspond to the output of `ica`, then we can consider that

$$\mathbf{M} = A\mathbf{H}$$

   In this case A is the mixing matrix which produces our observation $\mathbf{M}$. Compute the 2×2 matrix for this case. Submit A as `mixing_matrix.csv`.