Machine Learning for Signal Processing
Homework 2

Bhiksha Raj - Abelino Jimenez, Anurag Kumar, Yixuan Zhang, Davy Uwizera

Sections 1, 2 and 3 are questions of homework 2. Section 4 is matlab instruction which can help you with resolving problems. Section 5 is a submission guide.

1 Gender Classification

In this problem you have to implement a gender classifier from face images. To do so, you have been provided of a subset of the LFW data set. You must use eigenfaces and Adaboost to perform the classification.

1.1 Eigen Faces

In the directory `hw2materials/problem1` you can find 2 folders: `training` and `testing`. In each of these you can find two folders: `women` and `men`. In the folder `women` there are pictures of women faces, while in the folder `men` there are men faces. Each of these images is a 36×36 gray scale image.

1. Using the images in `training`, find the first eigen face using the women images. Repeat this for the men images. Plot both eigen faces. For each group (men and women) in the training set, compute the mean of the error reconstruction using the first eigen face from women. Repeat this using the principal eigen face from men. Report the 4 requested errors.

1.2 Adaboost

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

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

where h_t are weak classifiers and α_t the weights computed in the training phase.

2. Implement your own version of Adaboost. Write the function `adaboost` 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 used.
   - `X_te`: a $P \times M$ matrix, where each row corresponds to a data sample with M dimensions.

   Your objective is, using as training data `X_tr` and `Y_tr`, training an Adaboost classifier and making predictions for `X_te`. Then, the output of this function must be:

   - `pred_te`: a $P \times 1$ matrix, where each component can be either 1 or -1. The i-th component is the prediction for the i-th row of `X_te`.

This implementation will be graded based on the performance obtained in the following problem.

1.3 Gender Classification

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

Using the training set, you can compute K eigen faces using just the women pictures. You can repeat this using the men pictures only, having $2K$ eigen faces in total. Now, for any image, you can compute its projection to each of these eigen faces, obtaining $2K$ real values.

3. Considering $K = 10$, for every image compute the projection to each eigen face, and use these values as features for your classifier. Note that for each image you have to compute a vector with 20 components. To define `Y_tr`, consider 1 for women and -1 for men. Train your model using different values of `T` (10,50,100,150,200). Plot your accuracy both in the training and testing set as a function of T.

4. Repeat and report the previous step taking $K = 50$, 100, 150, 200. Is there any improvement?

Write a small report including the requested figures.

2 Non Negative Matrix Factorization

Non Negative Matrix Factorization is an extremely useful algorithm and has applications in several areas. In this problem you will use it to solve two problems. You will use and implement the multiplicative NMF algorithm described on slide 48 in lecture presentation.

2.1 Face Reconstruction

In class we saw that NMF can be used to reconstruct faces. In this problem you will implement NMF algorithm for facial reconstruction.

The homework materials for problem 2 contains a `.mat` file with training faces (`problem2/train_faces.mat`. You will use these faces to learn bases for faces. The faces have been taken from LFW dataset[1]. You can load this .mat file as `train_faces = load(path_to_mat_file)`. Each column of this matrix contains a face. Visualize some of the faces by reshaping it as `32 by 32` matrix and using `imshow` in matlab. In the same folder you will see `test_faces.mat` and `test_faces_cor.mat` which you will use for testing. These are in the same format as `train_faces.mat` .

Implement the following functions.

1. Implement a function `B = doNMF(inmat,K,niter,Binit,Winit)` which returns the bases for input data (train faces) in `inmat`. K is the number of bases, `niter` is number of iteration and `Binit, Winit` are initial values of B and W.

2. Implement a function `recon = NMFreco(inmat,B,niter,Winit)` which takes test faces matrix , bases B learned from above function, `niter` and `Winit`. It reconstructs the faces using B. Essentially, you want to learn W and reconstruct the faces as $B * W$.

2.1.1 Reconstruction 1

Reconstruct faces using the above two functions for `problem2/test_faces.mat`. Use $K = 100, 200, 300, 400, 500$. When learning bases, use `problem2/B[K].mat` and `problem2/W[K].mat` for initializing B and W for each K. When learning weights during reconstruction, use `problem2/Wtest[K].mat` to initialize your W. Use `niter=500` in all cases.

2.1.2 Reconstruction 2

In `problem2/test_faces_cor.mat`, a portion of the test faces have been corrupted. Visualize some of the faces. Repeat everything in above problem on this corrupted set.

2.1.3 What to submit and report ?

1. Your functions `doNMF()` and `NMFreco()` and the main script which does rest of the stuff. Your main script should be called `faceMain.m`. You are free to write any other function you want. Make sure that running `faceMain.m` gets the job done. If needed put additional instructions at the top of `faceMain.m`. You will find template scripts in the "problem2" directory.

2. Visualize some of the reconstructed faces. In your report show the original and corresponding reconstructed faces. Do it for all K. Do it for both corrupted and uncorrupted test faces.

3. Report mean reconstruction error for each K, mean over all test images. Error for each test face is the squared error $||\hat{F}_i - F_i||_2^2$. Do it for both corrupted and uncorrupted test faces.

[1] http://vis-www.cs.umass.edu/lfw/

2.2 Signal Separation

In this problem you will use NMF to separate a mixed signal into its component signals. Specifically, your goal is to separate music and speech sounds from a signal consisting of both.

The basic idea is that you will use NMF to first learn bases for speech and music. Then you will use the learned bases to separate out music and speech from a recording consisting of both.

Do the following processing steps.

1. In `problem2` folder of homework materials, you will find `speechf1.wav` file. Read speech signal and compute its spectrogram. You did this in previous homework. Use the stft function as before `spectrum = stft(s',2048,256,0,hann(2048))` followed by mag computation.
2. Do the same for `musicf1.wav` wav file.

Now using the spectrograms for music and speech you are first going to learn bases, B_m and B_s for speech. Use the function you wrote in previous problem. Set $K = 200$ and use `Bminit` and `Wminit` in `problem2` folder for initializing B and W for music. Similarly for speech use `Bsinit` and `Wsinit`. Use $niter = 250$

Now compute the spectrogram ($\mathrm{V}_{\mathrm{mixed}}$) and phase of the the mixed signal.

1. Implement a function `[speech_rec, music_rec] = separate_signals(`$\mathtt{V}_{\mathtt{mixed}}$`,`$\mathtt{B}_{\mathtt{music}}$`,`$\mathtt{B}_{\mathtt{music}}$`, niter)`. $\mathtt{V}_{\mathtt{mixed}}$ is spectrogram of mixed signal, $\mathtt{B}_{\mathtt{music}}$ is bases you learned for music, $\mathtt{B}_{\mathtt{speech}}$ is bases for speech. `speech_rec` is recovered speech spectrogram and `music_rec` is recovered music spectrogram.

Now using the phase for the mixed signal along with reconstructed spectrograms, reconstruct time domain music and speech signals. You did this in last homework as well. You can use the `stft()` function again to do this. Listen them. Do you think NMF does a decent job in separating signals.

2.2.1 What to submit and report ?

- Submit all scripts for this problem, including `separate_signals()` function. Your main script should be called `sigsepMain.m`. A template is available in "problem2" subdirectory.
- Submit learned bases B_m and B_s for music and speech. Save them as `.mat` files.
- Submit your time domain music and speech signals, which you obtained by decomposing the mixed signal.

3 Independent Component Analysis

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. Read the file `sample1.wav` and extract the audio signal s1. This should be a vector with 132203 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 M with 2 rows and 132203 columns.

2. 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.

3. Apply the function `ica` on the matrix M and using the matlab function `audiowrite`, save the components generated as `source1.wav` and `source2.wav` respectively. Dot forget ICA does not consider scale factors, so you may need to boost or decrease the resulting signal.

4. We have provided the magnitude of the spectrogram S_1 and S_2 (`hw2materials/problem3/spectrograms`), for the original signals. Compute the magnitude of the spectrogram $R1$ and $R2$ for your resulting signal and compare with the original ones. In particular, compute and report the following errors

$$error_1 = \min(\|S_1 - R_1\|_{\text{Frobenious}}, \|S_1 - R_2\|_{\text{Frobenious}})$$

and

$$error_2 = \min(\|S_2 - R_1\|_{\text{Frobenious}}, \|S_2 - R_2\|_{\text{Frobenious}})$$

Use the `stft` function provided in homework 1, considering 2048 sample windows and a shift of 256 samples.

4 Matlab Instructions

4.1 Reading in Images

You are given a collection of images of faces. These images are all dimension 36×36. From these you must compute the "best" eigen faces.

The matlab command to read an image is:

```
image = double(imread(imagefile));
```

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

4.2 Building A Matrix of Images

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 x ncolumns image into a single nrows*ncolums x 1 vector. In our "training" data every image is the same size (36 x 36) and no rescaling of images is necessary. 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
```

Since all images are the same size, you can get the size of the image from image1.

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

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

4.3 Computing Eigen faces

Eigen faces can be computed from Y, which is an (nrows*ncolumns) x nimages matrix. There are two ways to to compute eigen faces; one is using eigen analysis and the other is by singular value decomposition.

4.3.1 Eigen analysis

First compute the correlation matrix: `corrmatrix = Y * Y';` How large is the correlation matrix? You can compute eigen vectors and eigen values of the correlation matrix as:

```
[eigvecs, eigvals] = eig(corrmatrix);
```

The columns of the eigenvecs matrix (in matlab notation the i-th column is given by eigvecs(:,i)) are the eigen vectors. The diagonal entries of the eigvals matrix are the eigen values. You can confirm that corrmatrix = eigvecs * eigvals * eigvecs'. To learn more about eigen analysis and the eig() command, type "help eig" into matlab.

The intuition for what Eigen vectors and Eigen values really mean is roughly as follows: The eigen vectors are a number of "orthogonal" bases that can be used to compose every image in the collection. What do we mean by "orthogonoal"? Compute the dot product between any two eigen vectors: eigvecs(:,i)*eigvecs(:,j)' (note the apostrophe representing the transpose). If $i \neq j$, then the dot product will be 0. In other words, we cannot explain any part of the i-th eigen vector by the j-th eigen vector. As a consequence, the parts of any image that are explained by the i-th eigen vector cannot be explained by any other eigen vector.

If we composed every image in the collection as a linear combination of our eigen vectors: image = a1 eigvecs(:,1) + a2 eigvecs(:,2) + a3eigvecs(:,3)..., then ai indicates the contribution of the i-th eigen vector to the image. The i-th eigenvalue gives us the RMS value of ai. It tells us how much the i-th eigen vector contributes to the images, typically. The lower the i-th eigen value, the less the i-th eigen vector contributes to the composition of the images. If the i-th eigen value is zero, the i-th eigen vector does not contribute to the composition of the images at all!

You will find it instructive to plot the Eigen values: plot(diag(eigvals));. Matlab's eig() function returns eigen vectors in order of increasing importance. The first eigen vector is the least important. The last one is the most important. The plot of eigen values will also show this – the last eigen value is the largest. Note also that if you have k images in Y, no more than k Eigen values are non-zero. That is because you will not need more than k Eigen vectors to explain k images; the importance of the remaining Eigen vectors is zero.

If you have gone through the above exercise, you will quickly realize that it takes a long time to perform eigen analysis. If our images were larger (more pixels), it would take much much longer. One of the reasons is that eig() operates on a large correlation matrix. Another it that it computes all eigen values and eigen vectors, whereas we know that if we have k images in Y, only k of the Eigen vectors (and Eigen values) are relevant – the others have zero importance. The faster way to achieve the same end is via singluar value decomposition:

4.3.2 Singular Value Decomposition (SVD)

SVD can be performed in matlab as `[U,S,V] = svd(Y,0);`

Note that SVD is performed directly on Y and not on the correlation matrix. The "0" to the right states that we want a "thin" SVD. The thin SVD will give us an (nrows*ncolumns) x k matrix U, and a k x k diagonal matrix of singular values, S. The matrix V are the right singular values and we need not concern ourselves with it for this problem.

You can confirm that the k columns of U are the same as the final k columns of the Eigen vectors matrix returned by eig(corrmatrix), only in reverse order. i.e. the first column of U is the most important eigen vector, the second column is the second-most important Eigen vector and so on. The diagonal entries of S are also the square roots of the final k diagonal entries of the eigvals matrix returned by eig(), only in reverse order.

The gist of the above is that instead of running eigen analysis using eig(corrmatrix), you can get the desired Eigen vectors using svd(Y,0) and it will be much faster.

svd() still computes k eigen vectors and singular values. For our problem we only want one (or, for problem 2, some J) eigen vector. Obviously the effort expended in computing the rest of the eigen vectors is wasted. An even faster version of SVD is given by the svds() command in matlab which only computes exactly as many Eigen vectors as you want. You can learn more about these by typing help svd or help svds into matlab.

You can use any of these techniques to compute the Eigen faces. The first Eigen face will be used to detect faces in the group photograph.

The eigen face will be in the form of a nrows*ncolumns x 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).

5 Submission Guide

5.1 Gender Classification

Please submit the figures for this part in "Report_YourAndrewID.pdf" in your submission. You also need to save your figures as .jpg files in the folder "results".

The "problem1" subdirectory has a template file named `run_problem1_1.m`. Write your code into this script to solve part 1.1.

You can also find a file named `adaboost.m`. Fill this script to implement your version of Adaboost.

Finally, we can find a file named `run_problem1_3.m`. Fill it to solve part 1.3.

5.2 Non Negative Matrix Factorization

All submissions should be in "problem2" subdirectory.

5.2.1 Face Reconstruction

Follow the instructions in the problem. Submit codes and report accordingly.

5.2.2 Signal Separation

Submit Bs.mat and Bm.mat in results directory inside "problem2". Submit recovered music signal as `music_rec.wav` and recovered speech as `speech_rec.wav` inside results directory.

5.3 Independent Component Analysis

Please, write in "Report_YourAndrewID.pdf" a small description of your implementation and your impression about your results.

In the subdirectory "problem3" has file name `ica.m`. Please, fill it with your implementation of ica.

You also can find a file named `run_problem3.m`. Please, fill it in order to separate the mixed signal provided.