Machine Learning for Signal Processing (Residential) (ENGR-E 511; CSCI-B 590) Homework 3

Due date: Mar. 15, 2024, 23:59 PM (US Eastern)

Instructions

- Submission format: Jupyter Notebook or MATLAB Livescript) + HTML
  - Your notebook should be a comprehensive report, not just a code snippet. Mark-ups are mandatory to answer the homework questions. You need to use LaTeX equations in the markup if you're asked.
  - Google Colab is the best place to begin with if this is the first time using iPython notebook. No need to use GPUs.
  - Download your notebook as an .html version and submit it as well, so that the AIs can check out the plots and audio. Here is how to convert to html in Google Colab.
  - Meaning you need to embed an audio player in there if you're asked to submit an audio file.
- Avoid using toolboxes.

P1: Instantaneous Source Separation [3 Points]

1. From `x_ica_1.wav` to `x_ica_20.wav` are 20 recordings of my song, `Homework 3`. Each recording has N time domain samples. In this music there are K **unknown number** of musical sources played at the same time. This can be seen as a situation where the source was mixed up by multiplying a $20 \times K$ mixing matrix $\boldsymbol{A}$ to the K sources, creating the 20-channel mixture:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_{20}(t) \end{bmatrix} = \boldsymbol{A} \begin{bmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_K(t) \end{bmatrix} \tag{1}$$

2. In this question we use ICA to separate the K independent signals from each instrument source.

3. First, we need to isolate a number of recordings same as the number of sources for ICA. And if you listen carefully, you'll notice the actual number of sources (instruments) are much less than 20. So, let's perform PCA with the whitening option. Apply your PCA algorithm on your data matrix $\boldsymbol{X}$, a $20 \times N$ matrix. Don't forget to whiten the data. Make a decision as to how many dimensions to keep, which will correspond to your K. Hint: take a very close look at your eigenvalues, because there is a rather small element that is useful for separation.

4. On your whitened/dimension reduced data matrix $\boldsymbol{Z} \in \mathbb{R}^{K \times N}$, apply ICA. At every iteration of the ICA algorithm, use these as your update rules:

$$\begin{aligned} \Delta \boldsymbol{W} &\leftarrow \left(N\boldsymbol{I} - g(\boldsymbol{Y})f(\boldsymbol{Y})^\top\right)\boldsymbol{W} \\ \boldsymbol{W} &\leftarrow \boldsymbol{W} + \rho \Delta \boldsymbol{W} \\ \boldsymbol{Y} &\leftarrow \boldsymbol{W}\boldsymbol{Z} \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{W} &: \text{The ICA unmixing matrix you're estimating} \\ \boldsymbol{Y} &: \text{The } K \times N \text{ source matrix you're estimating} \\ \boldsymbol{Z} &: \text{Whitened/dim reduced version of your input (using PCA)} \\ g(x) &: \tanh(x) \\ f(x) &: x^3 \\ \rho &: \text{learning rate} \\ N &: \text{number of samples} \end{aligned}$$

5. Enjoy your music. Submit K separated sources (by embedding audio player to your .HTML version), source code, and the convergence graph.

6. Implementation notes: Depending on the choice of the learning rate, the convergence of the ICA algorithm varies. But I always see a convergence under 50s.

P2: Single-channel Source Separation [3 Points]

1. You can use a variety of error functions for the NMF algorithm. An interesting one is to see the matrices as a probabilistic distribution though not normalized. I actually prefer this one than the one I introduced in L6 S35. From this, you can come up with this error function:

$$\mathcal{E}(\boldsymbol{X}||\boldsymbol{W}\boldsymbol{H}) = \sum_{i,j} X_{i,j} \log \frac{X_{i,j}}{\hat{X}_{i,j}} - X_{i,j} + \hat{X}_{i,j}, \tag{2}$$

where $\hat{\boldsymbol{X}} = \boldsymbol{W}\boldsymbol{H}$. I wouldn't bother you with a potential question that might ask you to derive the update rules. Instead, I'll just give them away here:

$$\boldsymbol{W} \leftarrow \boldsymbol{W} \odot \frac{\frac{\boldsymbol{X}}{\boldsymbol{W}\boldsymbol{H}}\boldsymbol{H}^\top}{\boldsymbol{1}^{F\times T}\boldsymbol{H}^\top} \tag{3}$$

$$\boldsymbol{H} \leftarrow \boldsymbol{H} \odot \frac{\boldsymbol{W}^\top \frac{\boldsymbol{X}}{\boldsymbol{W}\boldsymbol{H}}}{\boldsymbol{W}^\top \boldsymbol{1}^{F\times T}}. \tag{4}$$

Note that $\boldsymbol{1}^{F\times T}$ is a matrix full of ones, whose size is $F \times T$, where F and T are the number of rows and columns of your input matrix $\boldsymbol{X}$, respectively. Please note that, once again, we have multiplicative update rules. Therefore, once you initialized your parameter matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ with nonnegative random values, you can keep their nonnegativity as long as your input matrix $\boldsymbol{X}$ is nonnegative as well.

2. `trs.wav` is a speech signal of a speaker. Load it and convert it into the time-frequency domain by using STFT with a frame size of 1024 samples with 50% overlap. Use Hann windows. This will give 513×990 complex-valued matrix (You could get slightly more or fewer columns than mine depending on your STFT setup).

3. Take the magnitudes of this matrix. Let's call these magnitudes $\boldsymbol{S}$, which is nothing but a matrix of nonnegative elements. Learn an NMF model out of this, such as $\boldsymbol{S} \approx \boldsymbol{W_S H_S}$, using the update rules (3) and (4). You know, $\boldsymbol{W_S}$ is a set of basis vectors. If you choose to learn this NMF model with 30 basis vectors, then $\boldsymbol{W_S} \in \mathbb{R}_+^{513\times30}$, where $\mathbb{R}_+$ is a set of nonnegative real numbers. You're going to use $\boldsymbol{W_S}$ for your separation.

4. Repeat this process for your other training signal `trn.wav`. Learn another NMF model from `trn.wav`, which is another training signal for your noise. From this get $\boldsymbol{W_N}$.

5. `x_nmf.wav` is a noisy speech signal made of the same speaker's different speech and the same type of noise you saw. By using our third NMF model, we're going to denoise this one. Load this signal and convert it into a spectrogram $\boldsymbol{X} \in \mathbb{C}^{513\times131}$. Let's call its magnitude spectrogram $\boldsymbol{Y} = |\boldsymbol{X}| \in \mathbb{R}_+^{513\times131}$. Your third NMF will learn this approximation:

$$\boldsymbol{Y} \approx [\boldsymbol{W_S W_N}]\boldsymbol{H}. \tag{5}$$

What this means is that for this third NMF model, instead of learning new basis vectors, you reuse the ones you trained from the previous two models as your basis vectors for testing: $\boldsymbol{W} = [\boldsymbol{W_S W_N}]$. As you are very sure that the basis vectors for your test signal should be the same with the ones you trained from each of the sources, you initialize your $\boldsymbol{W}$ matrix with the trained ones and don't even update it during this third NMF. Instead, you learn a whole new $\boldsymbol{H} \in \mathbb{R}_+^{60\times131}$ that tells you the activation of the basis vectors for every given time frame. Implementation is simple. Skip the update for $\boldsymbol{W}$. Update $\boldsymbol{H}$ by using $\boldsymbol{W} = [\boldsymbol{W_S W_N}]$ and $\boldsymbol{Y}$. Repeat. Note: You're doing a lot of element-wise division, so be careful not to divide by zero. To prevent this, I tend to add a very very small value (e.g. 10^{-20}) to every denominator element in the update rule.

6. Because $\boldsymbol{W_S H}_{(1:30,:)}$ can be seen as the speech source estimate, you can also create a masking matrix out of it:

$$\bar{\boldsymbol{M}} = \frac{\boldsymbol{W_S H}_{(1:30,:)}}{\boldsymbol{W_S H}_{(1:30,:)} + \boldsymbol{W_N H}_{(31:60,:)}} = \frac{\boldsymbol{W_S H}_{(1:30,:)}}{[\boldsymbol{W_S W_N}]\boldsymbol{H}}. \tag{6}$$

Use this masking matrix to recover your speech source, i.e. $\hat{\boldsymbol{S}} = \bar{\boldsymbol{M}} \odot \boldsymbol{X}$. Note that you are multiplying a nonnegative masking matrix $\bar{\boldsymbol{M}}$ to a complex matrix $\boldsymbol{X}$, whose result is also complex-valued. Invert back to the time domain. Listen to it to see if the noise is suppressed. Submit the audio result and source code.

P3: Motor Imagery [3 Points]

1. `eeg.mat` has the training and testing samples and their labels. Use them to replicate my BCI classification experiments in Module 5 (not the entire lecture, but from S3 to S8 and S37). But, instead of PCA dimension reduction, we're going to use locality sensitive hashing to extract binary features. Also, instead of naïve Bayes, we'll do kNN classification.

2. For the kNN classification, for every test example you have to find the K nearest neighbors from 112 training samples. You're lucky. This kNN search is not a big deal, because your training set is tiny. However, as your professor I have to help you guys be prepared for your future big data projects. So, I'm giving you this homework assignment that will help you come up with a speed-up technique for the kNN search. It's not that complicated.

3. Come up with a random projection matrix $\boldsymbol{A} \in \mathbb{R}^{L \times M}$, where M denotes the number of dimensions of your data samples (5 rows of your magnitude STFT, vectorized), 255 [1], and L is the new dimension after this random projection. So, you're replacing PCA with this random projection. Note that L doesn't always have to be smaller than M. Although $\boldsymbol{A}$ is a matrix with random values, you should make sure that the row vectors (the projection vectors) should be unit vectors, i.e. $\|\boldsymbol{A}_{i,:}\|_2 = 1$. Make sure of it.

4. Do the random projection and take the sign of the result (element-wise), so that your random projection produces a bunch of +1 's and -1 's:

$$\boldsymbol{Y} = \text{sign}\left(\boldsymbol{A}\boldsymbol{X}_{1:M,:}\right) \tag{7}$$

5. Do your kNN classification using these binary version of your data samples. Note that you can compare the test bit string with the bit strings of the training data by using the Hamming distance, instead of the Eucleadian distance between the original real-valued vectors. You know your computer prefers binary values, right? This way you can speed up the comparison, and eventually the kNN search (although you might not be able to see the actual speed-up due to the too small dataset size and your script language that doesn't exploit binary variables). Report your classification results with different choice of K and L. I think a table of accuracies will be great. You don't have to submit all the intermediate plots. What I need to see is the accuracies and your source code.

6. Compare your results with mine in M5 S37. You're classifier is based on binary variables and bitwise operations, so you must have expected a big performance drop. How do you like it eventually? Report your classification accuracy. Submit your code and your discussion.

P4: kNN Source Separation [3 Points]

1. Prepare your $\boldsymbol{S}$ and $\boldsymbol{N}$ matrices by using the procedure in 2.2 – 2.4, but without NMF, i.e. they are the input magnitude spectrogram to your NMF. This time, let's create another matrix $\boldsymbol{G}$, the magnitude spectrogram of trs.wav+trn.wav. Also, let's create another matrix called Ideal Binary Masks (IBM), $\boldsymbol{B}$ defined as follows:

$$\boldsymbol{B}_{f,t} = \begin{cases} 1 & \text{if } \boldsymbol{S}_{f,t} \geq \boldsymbol{N}_{f,t} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

My hope is that $\boldsymbol{S} \approx \boldsymbol{B} \odot \boldsymbol{G}$. This sounds a bit too rough, but it actually works well, because if you take a look at the local bins, they are dominated by one of the sources, not really all of them due to the sparsity of the sources in the TF domain. If you can't believe in me, go

[1] Note that this number comes from the calculation [# channels] × [# frames] × [# subbands], while # frames can slightly vary depending on your STFT implementation. Mine resulted in $3 \times 17 \times 5 = 255$

ahead and convert $\boldsymbol{S}$ back to the time domain (with proper phase information) to check out the sound, although I don't require it.

2. Load `x_nmf.wav` and convert it into $\boldsymbol{X}$ and $\boldsymbol{Y}$ like in 2.5. We're not doing NMF on this though. This time, we compare each column of $\boldsymbol{Y}$ with all columns in $\boldsymbol{G}$ to find out k nearest neighbors. Then, for t-th given test frame, we have a list of indices to the k nearest frames in $\boldsymbol{G}$, e.g. $\{i_1, i_2, \cdots, i_k\}$, where $i_1, i_2, \cdots, i_k \in \{1, \cdots, 990\}$. For the comparison, I used Euclidean distance for no good reason, but you can choose a different (more proper) one if you want.

3. What we want to do with this is to predict the IBM of the test signal, $\boldsymbol{D} \in \mathbb{B}^{513\times 131}$. By using the nearest neighbor indices, I can collect the IBM vectors from $\boldsymbol{B}$, i.e. $\{\boldsymbol{B}_{:,i_1}, \boldsymbol{B}_{:,i_2}, \cdots, \boldsymbol{B}_{:,i_k}\}$. Since $\boldsymbol{Y}_{:,t}$ is very similar to $\{\boldsymbol{G}_{:,i_1}, \boldsymbol{G}_{:,i_2}, \cdots, \boldsymbol{G}_{:,i_k}\}$, I hope that my $\boldsymbol{D}_{:,t}$ should be similar to $\{\boldsymbol{B}_{:,i_1}, \boldsymbol{B}_{:,i_2}, \cdots, \boldsymbol{B}_{:,i_k}\}$. How do we consolidate them though? Let's try median of them, because that'll give you one if one is the majority, and vice versa:

$$\boldsymbol{D}_{:,t} = \text{median}\left(\{\boldsymbol{B}_{:,i_1}, \boldsymbol{B}_{:,i_2}, \cdots, \boldsymbol{B}_{:,i_k}\}\right). \tag{9}$$

4. Repeat this for all your test frames (131 frames in my STFT) and predict the full $\boldsymbol{D}$ matrix. Recover the complex-valued spectrogram of your speech by masking the input: $\hat{\boldsymbol{S}}_{\text{test}} = \boldsymbol{D} \odot \boldsymbol{X}$. Convert it back to the time domain. Listen to it. Is the noise suppressed? Submit this wave file as well as your code.