Machine Learning for Signals (CS 545)
Homework 2

Due date: Oct. 13, 23:59 PM (Central)

Instructions

- Please don't submit your homework.
- The solution will be distributed on the due date.
- Please still do your homework, as the quizzes and exams will be based on the homework assignments.
- Unless mentioned otherwise, you are expected to implement all the functions instead of using existing toolboxes, e.g., from `sklearn`.
- Avoid using toolboxes.

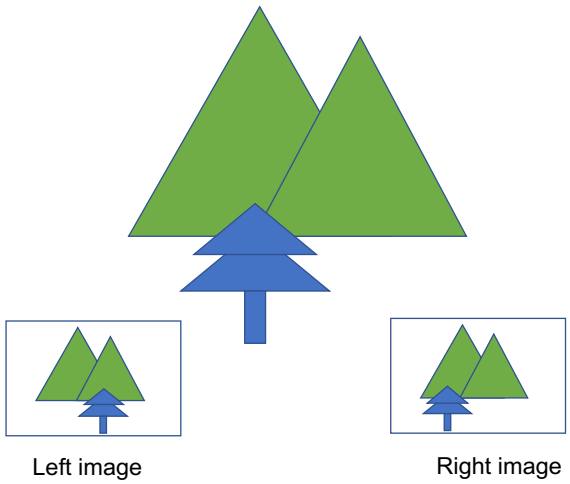

Figure 1: The tree is closer than the mountain. So, from the left camera, the tree is located on the right-hand side, while the right camera captures it on the left-hand side. On the contrary, the mountain in the back does not have this disparity.

P1: Parallax

1. You live on a planet far away from the Earth. Your solar system belongs to a galaxy that is about to merge with another galaxy (it is not rare in outer space, but don't worry, the merger takes a few billion years). Anyhow, because of this merger, in your deep sky you see lots of stars from your galaxy as well as the other stars in the other neighboring galaxy. Of course, you don't know which one is from which galaxy by eyeballing them, though.

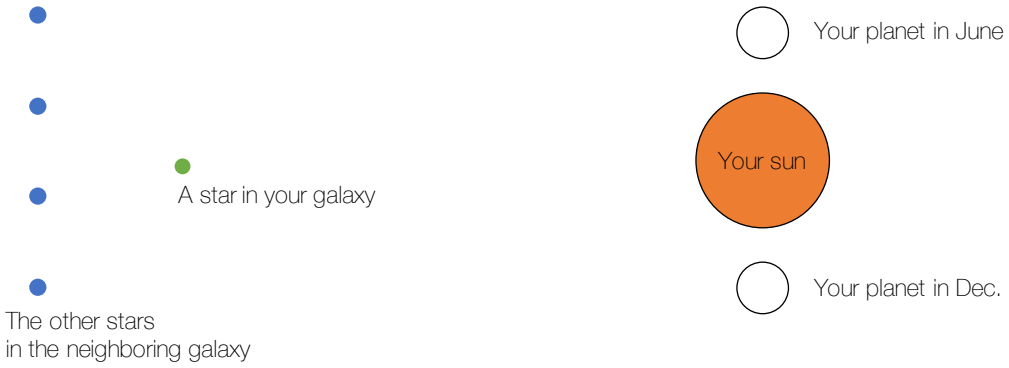

Figure 2: You take two pictures of the same area of the sky in June and December, respectively. Because of the pretty big movement of your planet due to its revolution, you can see that the close-by stars oscillate more in the two pictures than the far-away ones, just like the tree and the mountain in Figure 1.

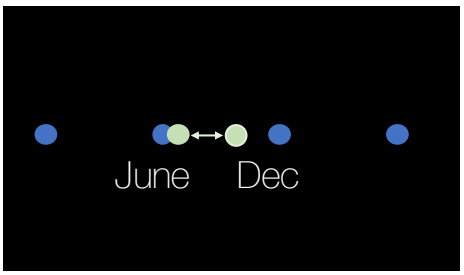

Figure 3: The oscillation of the close star (green) in the two pictures. Note that the other stars (blue) don't move much.

2. You are going to use a technique called "parallax" to solve this problem. It's actually similar to the computer vision algorithm called "stereo matching" that stereophonic cameras use to find out the 3D depth information from the scene. That's actually why we humans can recognize the distance of a visual object (we have two eyes). See Figure 1 for an example.

3. Let's get back to your remote planet. On your planet, parallax works by taking a picture of the deep sky in June and another one in December (yes, you have 12 months on your planet, too). If you take a picture of the deep sky, you see the stars nearby (i.e., the ones in your galaxy) change their position much more in the two pictures, while the stars far away (i.e., the ones in the neighboring galaxy) change their position less. See Figure 2 and 3.

4. `june.png` and `december.png` are the two pictures you took for this parallax project. Take a look at these deep sky images for your peace of mind.

5. In theory, you need to apply a computer vision technique called "non-maximum suppression," with which you can identify the position of all the stars in the two pictures. In theory, for each of the stars in `june.png`, you look for its position in `december.png` by scanning a star

in the same row (because the stars always move to the right). But I'm going to delegate this part to a proper computer vision class.

6. Instead, I provide you with the (x, y) coordinates of all the stars in the two pictures. `june.npy` and `december.npy` contain the positions of the stars in the two pictures. Each row has two coordinates, x-coordinate, and y-coordinate, and there are 2,700 such rows in each matrix, each of which is for a particular star.

7. If you take the first column vectors of the two matrices (i.e., the x-coordinates of the stars), you can subtract the ones in June from the ones in December to find out their *disparity*, or the amount of oscillation, which is a vector of 2,700 elements. Draw a histogram out of these disparity values to gauge if you can see two clusters there.

8. Perform k-means clustering on this disparity dataset. Find out the cluster means and report the values. Which one of the two clusters do you think corresponds to the stars in your galaxy, and which is for the other galaxy? Why do you think so?

9. Do you like the Kmeans clustering result? If not, why not?

10. Write up your own GMM clustering code and then perform clustering on the same disparity values in $\boldsymbol{D}$. The posterior probability will give you the membership of each value to one of the clusters. Compared to the Kmeans clustering result, how would you like about the new result from GMM?

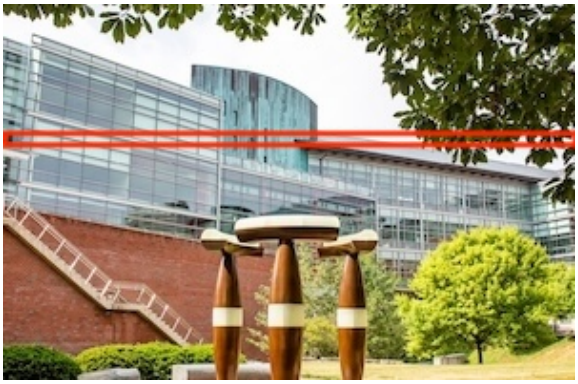

Figure 4:

P2: PCA or DCT

1. Load the picture of the Siebel Center and divide the 3D array ($300 \times 197 \times 3$) into the three channels.

2. Collapse them into one channel, i.e., $300 \times 197 \times 3$, so it becomes a gray-scale image. You can just average the pixels over the channel dimension. Let's call this image $\boldsymbol{X}$.

3. Randomly choose a block of 8 consecutive (entire) rows from $\boldsymbol{X}$, e.g. $\boldsymbol{X}^R_{(113:120,1:197)}$ (See the red box in Figure 4). This will be a matrix of 8×197.

4. Subtract the mean and calculate the covariance matrix, which will be an 8×8 matrix. Do eigendecomposition and extract eight eigenvectors, each of which has eight dimensions. Yes, you did PCA. Imagine that you convert the original 8×197 matrix into the other space using the learned eigenvectors. For example, if your eigenvector matrix is $\boldsymbol{W}$, then $\boldsymbol{W}^\top \boldsymbol{X}$ will do it. Plot your $\boldsymbol{W}^\top$ and compare it to the DCT matrix shown in L04-S21. Similar?

5.

6. We just saw that PCA might be able to replace DCT. But, it seems to depend on the quality of PCA. One way to improve the quality is to increase the size of your data set, so that you can start from a good sample covariance matrix. To do so, let's collect more rows from the data. Select nine more 8×197 blocks randomly and concatenate all of them horizontally, which will give you a new matrix of 8×1970. Do PCA on this. Any better?

7. How about concatenating 1,000 such blocks?

8. Think about the pros and cons of PCA and DCT.

P3: Instantaneous Source Separation

1. As you might have noticed from my long hair, I've got a rock spirit. However, for this homework, I dabbled with composing a jazzy piece of music. The title of the song is rather boring, though: `Homework 2`.

2. From `x_ica_1.wav` to `x_ica_20.wav` are 20 recordings of my song, `Homework 2`. Each recording has N time domain samples. In this music there are K unknown number of musical instruments (i.e., sources) played at the same time. In other words, as I wanted to disguise the number of sources, I created unnecessarily many recordings of this simple music. This can be seen as a situation where the source was mixed up with a $20 \times K$ mixing matrix $\boldsymbol{A}$ to the K sources to create the 20-channel mixture:

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

3. But, as you've learned how to do source separation using ICA, you should be able to separate them out into K clean speech sources.

4. First, you don't like the fact that there are too many recordings for this separation problem because you have a feeling that the number of sources is a lot smaller than 20. So, you decide to do a dimension reduction before you actually go ahead and do ICA. For this, you choose to perform PCA with a 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 that will correspond to your K. Hint: take a very close look at your eigenvalues.

5. On your whitened/dimension reduced data matrix $\boldsymbol{Z}$ $(K \times N)$, 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

$\boldsymbol{W}$: The ICA unmixing matrix you're estimating

$\boldsymbol{Y}$: The $K \times N$ source matrix you're estimating

$\boldsymbol{Z}$: Whitened/dim reduced version of your input (using PCA)

$g(x)$: $\tanh(x)$

$f(x)$: x^3

ρ : learning rate

N : number of samples

6. Enjoy your separated music stored in $\boldsymbol{Y}$.

7. Implementation notes: Depending on the choice of the learning rate, the convergence of the ICA algorithm varies. But I always see the convergence in from 5 sec to 90 sec on Google Colab.

P4: Single-channel Source Separation [2 points]

1. `trs.wav` is a speech signal of a speaker. Convert this signal into a spectrogram and take its magnitudes. Let's call this magnitude spectrogram $\boldsymbol{S}$. If you discard the complex conjugates, your $\boldsymbol{S}$ is a 513×990 matrix (the exact number of time frames can vary depending on your choice of STFT function).

2. Learn an NMF model out of this such as $\boldsymbol{S} \approx \boldsymbol{W_S}\boldsymbol{H_S}$. You know, $\boldsymbol{W_S}$ is a set of basis vectors. If you choose to learn this NMF model with 30 basis vectors, then $\boldsymbol{W_S} \in \mathbb{R}_+^{513\times 30}$, where $\mathbb{R}_+$ is a set of nonnegative real numbers. You're going to use $\boldsymbol{W_S}$ for your separation.

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

4. `x_nmf.wav` is a noisy speech signal made of the same speaker's different speech utterance and the same type of noise with `trn.wav`. By using our third NMF model, we're going to denoise this one. Load this signal and convert it into a spectrogram $\boldsymbol{X} \in \mathbb{C}^{513\times 131}$. Let's call its magnitude spectrogram $\boldsymbol{Y} = |\boldsymbol{X}| \in \mathbb{R}_+^{513\times 131}$. Your third NMF will learn this approximation:

$$\boldsymbol{Y} \approx [\boldsymbol{W_S}\boldsymbol{W_N}]\boldsymbol{H}. \tag{2}$$

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: $\boldsymbol{W} = [\boldsymbol{W_S}\boldsymbol{W_N}]$. As you are very sure that the basis vectors for your test signal should be the same as 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 $\boldsymbol{H} \in \mathbb{R}_{+}^{60\times 131}$ that tells you the activation of the basis vectors for a given time frame. Implementation is simple. Skip the update for $\boldsymbol{W}$. Update $\boldsymbol{H}$ by using $\boldsymbol{W} = [\boldsymbol{W_S} \boldsymbol{W_N}]$ and $\boldsymbol{Y}$. Repeat.

5. You can think of $\boldsymbol{W_S H}_{(1:30,:)}$ as an estimation of the speech spectrogram buried in the test signal. That being said, in order to convert it back to the time domain, you need its corresponding phase information, which you are missing. Instead, although it might not be perfect, the phase matrix of the test signal, i.e., $\angle \boldsymbol{X} = \frac{\boldsymbol{X}}{\boldsymbol{Y}}$, can be used to recover the complex-valued spectrogram of the speech source as follows:

$$\boldsymbol{s}_{\text{test}} \approx \text{iSTFT}(\angle \boldsymbol{X} \odot \boldsymbol{W_S H}_{(1:30,:)}) \tag{3}$$