Machine Learning for Signal Processing (Residential)
(ENGR-E 511; CSCI-B 590)
Homework 2

Due date: Feb. 23, 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: White Noise [3 points]

1. Have you ever wondered what it means by "white" noise? It's actually from the light. When the light is the sum of all visible frequencies, then it looks white to human eyes. If you pass the light through a prism, then you all of a sudden see the rainbow colors, so called a "spectrum." Yes, the prism does an analogue version of the Fourier transform.

2. So, even if we don't see the sound we listen to, if the signal consists of too many sinusoids with different frequencies, it sounds "white." I know, I know, it doesn't make sense.

3. You may also want to note that the sample distribution of a white noise signal looks like a Gaussian distribution, which is not news to us because we all know the central limit theorem.

4. `x.wav` is a speech signal contaminated by white noise. As I haven't taught you guys how to properly do speech enhancement yet, you're not supposed to know a machine learning-based solution to this problem (don't worry I'll cover it soon). Instead, you did learn how to do STFT, so I want you to at least manually erase the white noise from this signal to recover the clean speech source. For some reason, we know that the white noise added to the signal doesn't change its volume over time. So, what we're going to do is to listen to the sound and eyeball the spectrogram to find out the frames only with white noise. Then, we will build our simple noise model, with which we will suppress the noise in the other speech-plus-noise frames.
   (Note: don't forget to turn off the sampling rate option `sr=None` if you use `librosa.load`).

5. First off, create a DFT matrix $\boldsymbol{F}$ using the equation shown in M02-L01-S11 and S12. You'll of course create a $N \times N$ complex matrix, but if you see its real and imaginary versions separately, you'll see something like the ones in M02-L01-S14 (the ones in the slide are 20×20, i.e. $N = 20$). For this problem let's fix $N = 1024$.

6. Prepare your data matrix $\boldsymbol{X}$. You extract the first frame of N samples from the input signal, and apply a Hann window[1]. What that means is that from the definition of Hann window, you create a window of size N and element-wise multiply the window and your N audio samples. Place it as your first column vector of the data matrix $\boldsymbol{X}$. Move by $N/2$ samples. Extract another frame of N samples and apply the window. This goes to the second column vector of $\boldsymbol{X}$. Do it for your third frame (which should start from $(N+1)$'th sample, and so on. Since you moved just by the half of the frame size, your frames are overlapping each other by 50%. (Note: this time it's okay to use the toolbox to calculate Hann windows.)

7. Apply the DFT matrix to your data matrix, i.e. $\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{X}$. This is your spectrogram with complex values. See how it looks like (by taking magnitudes and plotting). For example, you can use `imshow` in `matplotlib`.

8. In this spectrogram, identify frames that are only with noise[2]. For example the ones at the end of signal would be a good choice. Take a sample mean of the chosen column vectors (the original magnitudes, not the exponentiated ones), e.g. $\boldsymbol{M} = \frac{1}{|\mathcal{C}_{\text{noise}}|} \sum_{i \in \mathcal{C}_{\text{noise}}} |\boldsymbol{Y}_{:,i}|$, where $\mathcal{C}_{\text{noise}}$ is the set of chosen frames and $|\mathcal{C}_{\text{noise}}|$ is the number of frames. This is your noise model.

9. Subtract $\boldsymbol{M}$ out of all the magnitude spectra, $|\boldsymbol{Y}|$. This will give you some residual magnitudes with suppressed noise. Be careful with negative values: you don't want them in your "magnitude" spectra. One quick method to remove them is to turn them into zeros. Get the original phase from the input spectrogram, i.e. $\boldsymbol{Y}/|\boldsymbol{Y}|$ (element-wise division), and multiply each of the phase values by the corresponding cleaned-up magnitude to recover the complex-valued spectra of the estimated clean speech.

10. Multiply the inverse DFT matrix, which you can also create by using the equation in S12. Let's call this $\boldsymbol{F}^*$. Since it's the inverse transform, $\boldsymbol{F}^*\boldsymbol{F} \approx I$ (you can check it, although the off diagonals might be a very small number rather than zero). You multipy this matrix to your spectrogram, which is with suppressed white noise, to get back to the recovered version of your data matrix, $\hat{\boldsymbol{X}}$. In theory this should give you a real-valued matrix, but you'll still see some imaginary parts with a very small value. Ignore them by just taking the real part. Reverse the procedure in 1.6 to get the time domain signal. Basically it must be a procedure that transpose every column vector of $\hat{\boldsymbol{X}}$ and overlap-and-add the right half of t-th row vector with the left half of the $(t+1)$-th row vector and so on. Listen to the signal to check if the white noise is suppressed.

[1] `https://en.wikipedia.org/wiki/Hann_function`
`https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.get_window.html`

[2] Depending on the plotting function you use, it's possible that you can't really "see" the white noise. It's because your white noise is not loud enough. What you can do to better visualize this spectrogram is to exaggerate the small magnitudes while suppress the large ones. For example, I can visualize $|\boldsymbol{Y}|^{0.5}$ instead of $|\boldsymbol{Y}|$, where exponentiation is element-wise. Don't worry about this visualization issue if you can see the white noise-only frames from your spectrogram.

11. Submit your code and the denoised audio file. Do NOT use any STFT functions you can find in toolboxes.

P2: DCT and PCA [3 points]

1. `s.wav` is a recording of Prof. K's voice. Load it. Randomly select 8 consecutive samples out of the 5,000,000 samples. This is your first column vector of your data matrix $\boldsymbol{X}$. Repeat this procedure 10 times. Then, the size of $\boldsymbol{X}$ is 8×10.

2. Calculate the covariance matrix out of this, whose size must be 8×8. Do eigendecomposition and extract 8 eigenvectors, each of which is with 8 dimensions. Yes, you just did PCA. Plot your $\boldsymbol{W}^\top$ matrix and compare it to the DCT matrix shown in M02-L01-S21. Similar? Submit your plot and code.

3. Create another data matrix with 100 samples, i.e. $\boldsymbol{X} \in \mathbb{R}^{8\times 100}$. Do PCA on this one. How about 1,000 samples? Can you see your PCA is getting better with larger datasets? Why do you feel that your PCA is getting better? Try to explain in comparison with the DCT matrix.

4. You just saw that PCA might be able to replace the pre-fixed DCT basis vectors. But, as you can see in your matrices, they are not the same. Discuss the pros and cons of PCA and DCT in your report.

P3: Parallax [3 points]

1. You live in a planet far away from the earth. Your solar system belongs to a galaxy, which is about to merge with another galaxy (it is not rare in the outer space, but don't worry, the merger takes a few billions of 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 though.

2. You are going to use a technique called "parallax" to solve this problem. It's actually very similar to the computer vision algorithm called "stereo matching" that stereophonic cameras are using 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. In your planet, parallax works by taking a picture of the deep sky in June and another one in December (yes, you have 12 months there, too). If you take a picture of the deep sky, you see the stars nearby (i.e. the ones in your galaxy) changes its position much more in the two pictures, while the starts far way (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. 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 right). But we will not do it in here.

5. Instead, you get the (x, y) coordinates of all the stars in the two pictures. `june.mat` and `december.mat` contain the positions of the stars in the two pictures. Each row has two

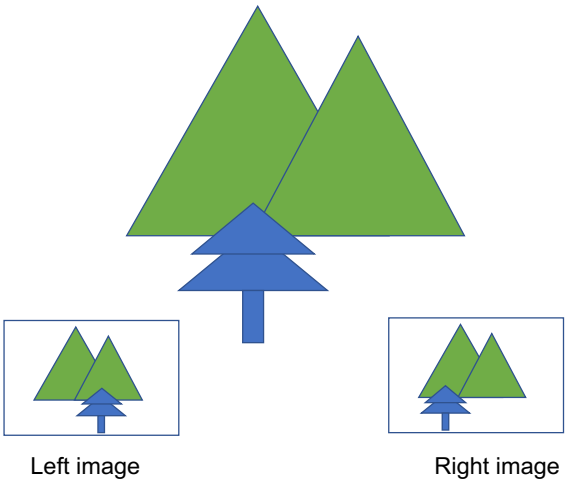

Figure 1: The tree is closer than the mountain. So, from the left camera, the tree is located on the right hand side, while the right camera captures it on the left hand side. On the contrary, the mountain in the back does not have this disparity.

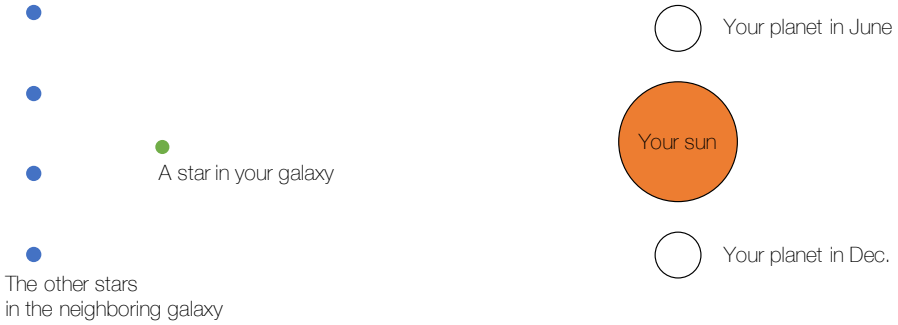

Figure 2: You take two pictures of the same area of the sky in June and December, respectively. Because of the pretty big movement of your planet due to its revolution, you can see that the close-by stars oscillate more in the two pictures than the far-away ones, just like the tree and the mountain in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

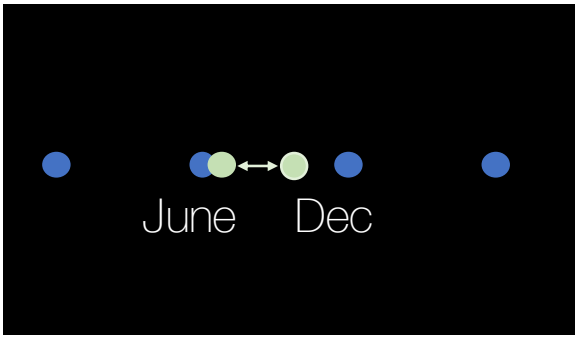

Figure 3: The oscillation of the close star (green) in the two pictures. Note that the other stars (blue) don't move much.

coordinates, x-coordinate and y-coordinate, and there are 2,700 such rows in each matrix, each of which is for a particular star. You can load the `.mat` files in python with the `scipy.io.loadmat(filepath)` function[3] [4].

6. 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 this disparity values to gauge if you can see two clusters there (pay attention to the bin size). Submit your histogram and answer in the report.

7. Write your **own** k-means clustering code, and apply on this disparity dataset. Find out the cluster means and report the values. Which one do you think corresponds to the stars in your galaxy and which is for the other galaxy? Why do you think so? Justify your answer in the report.

P4: GMM for Parallax [3 points]

1. Implement your own EM algorithm for GMM to propose an alternative solution to the parallax problem. Report your means. Explain which one you prefer between the k-means solution and the GMM results. Justify your answer.

[3] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html`
[4] Also, you might have to cast the value to `int16` in python to avoid overflowing.