Machine Learning for Signal Processing (ENGR-E 511) Homework 2

Due date: Feb. 23, 2020, 23:59 PM (Eastern)

Instructions

- No hand-written report
- Option 1: PDF + ZIP
  - A.pdf file generated from LaTeX
  - A.zip file that contains source codes and media files
- Option 2: IPython Notebook + 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.
  - Download your notebook as an .html version and submit it as well, so that the AIs can check out the plots and audio.
  - Meaning you need to embed an audio player in there if you're asked to submit an audio file
- Avoid using toolboxes.

P1: De-beeper [2 points]

1. `x.wav` is a speech signal contaminated by a beep sound. 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 beep sound from this signal to recover the clean speech source. If you have a perfect pitch, you might be able to know the pitch of the beep, but I assume that you don't. That's why you have to see the spectrogram to find out the beep frequency.

2. First off, create a DFT matrix $\boldsymbol{F}$ using the first equation in M02-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-S14 (the ones in the slide are 20×20). Determine your N, which is the frame size shown in M02-S17. For example, since the signal's sampling rate is 16kHz, if your N is 1600, your frequency resolution (the range a Fourier coefficient covers) will be 10Hz. If your N is large, you'll get finer frequency resolution and vice versa. Feel free to try out a few different choices.

3. Prepare your data matrix $\boldsymbol{X}$. You extract the first frame of N samples from the input signal, and apply a Hann window (or any other windows that can overlap-and-add to one). What that means is that from the definition of Hann window[1], 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%.

4. Apply the DFT matrix to your data matrix, i.e. $\boldsymbol{F}\boldsymbol{X}$. This is your spectrogram with complex values. See how it looks like (by taking magnitudes and plotting). Locate two thin horizontal lines. They are from the beep sound. Note that due to the conjugacy your spectrogram is mirrored vertically. The bottom half is a mirrored version of the top half in terms of their magnitudes, although their imaginary parts are with a different sign (complex conjugate). The spectrograms you've seen in class are the top half of a spectrogram, because the bottom half has no useful information (except for the flipped phase). This is why you see two beeper lines in your spectrogram. Anyway, locate them, and make the entire row zero.

5. Apply the inverse DFT matrix, which you can also create by using the equation in L4 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 apply this matrix to your spectrogram, which is free from the beep tones, 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 1.3 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 beep tones are gone.

6. Submit your code and the de-beepped audio file, or embed it into your notebook.

P2: Parallax [2 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 the 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," where stereophonic cameras find out the 3D depth information from the visual 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

[1] I allow you to use a pre-defined function to compute this window, but I encourage you to go ahead and implement one based on the simple equation: https://en.wikipedia.org/wiki/Window_function#Hann_and_Hamming_windows

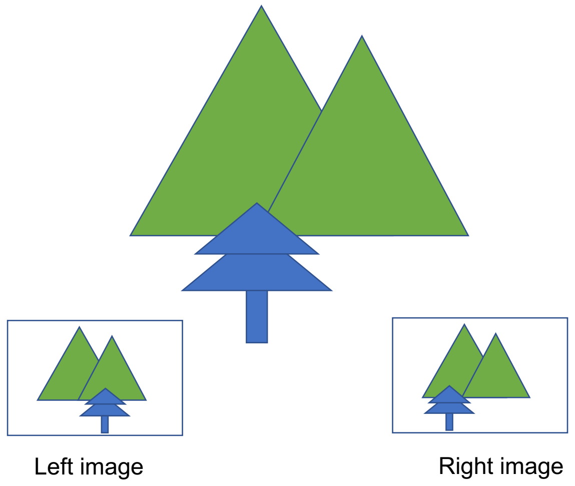

Figure 1: The tree is closer than the mountain. So, from the left camera, the tree is located on the right hand side, while the right camera captures it on the left hand side. On the contrary, the mountain in the back does not have this disparity.

you take a picture of the deep sky, you see the stars nearby (i.e. the ones in your galaxy) changes their 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, I'm going to save this part for Prof. Crandall so that he can use this topic in his homework assignment (if he wants to).
5. Instead, I provide you with 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 coordinates, x-coordinate and y-coordinate, and there are 2,700 such rows in each matrix, each of which is for a particular star.
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.
7. Perform k-means clustering 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.
8. Note that you're not supposed to use someone else' implementation. Write up your own code for k-means clustering.

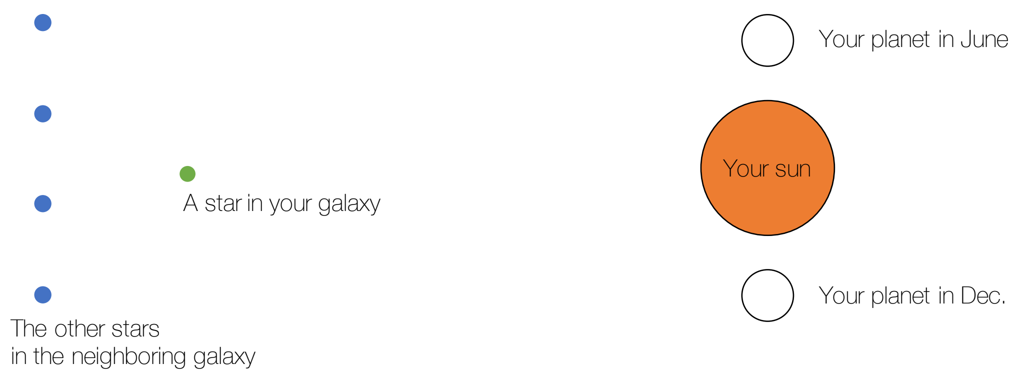

Figure 2: You take two pictures of the same area of the sky in June and December, respectively. Because of the pretty big movement of your planet due to its revolution, you can see that the close-by stars oscillate more in the two pictures than the far-away ones, just like the tree and the mountain in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

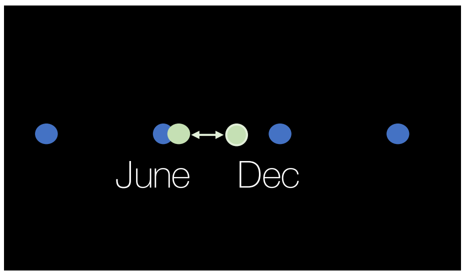

Figure 3: The oscillation of the close star (green) in the two pictures. Note that the other stars (blue) don't move much.

9. One tip for implementing k-means clustering from scratch is to make sure that each cluster contains at least one sample during the iterative updates. It's to avoid any trivial solution, e.g. the algorithm learns a gigantic cluster that contains everything.

P3: GMM for Parallax [2 points]

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

P4: DCT and PCA [2 points]

1. I like walking in the B-Line trail (although it doesn't mean that I have time to walk there frequently). `IMG_1878.JPG` is the photo I took there. Load it and divide the 3D array ($1024 \times 768 \times 3$) into the three channels. Let's call them $\boldsymbol{X}^R$, $\boldsymbol{X}^G$, and $\boldsymbol{X}^B$.

2. Randomly choose a block of 8 consecutive (entire) rows from $\boldsymbol{X}^R$, e.g. $\boldsymbol{X}^R_{(113:120,1:768)}$ (See the red box in Figure 4). This will be a matrix of 8×768. Collect another 2 such blocks, each of which starts from a randomly chosen first row position. Move on to the green channel and extract another three 8×768 blocks. Blue channel, too. You collected 9 blocks from all three channels. Now, concatenate all of them horizontally. This will be a matrix of 8×6912 pixels. Let's call it $\boldsymbol{R}$. Subtract the mean and calculate the covariance matrix, which will be an 8×8 matrix. Do eigendecomposition (feel free to use a toolbox) and extract 8 eigenvectors, each of which is with 8 dimensions. Yes, you did PCA. Imagine that you convert the original 8×6912 matrix into the other space using the learned eigenvectors. For example, if your eigenvector matrix is $\boldsymbol{W}$, than $\boldsymbol{W}^\top \boldsymbol{R}$ will do it. Plot your $\boldsymbol{W}^\top$ and compare it to the DCT matrix shown in M02-S21. Similar? Submit your plot and code.

3. 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, go back to the procedure in 4.2. But, this time increase the total number of blocks to 90 (30 blocks per channel). Note that each block is with 8×768 pixels once again. See if the eigenvectors are better looking (submit the plot).

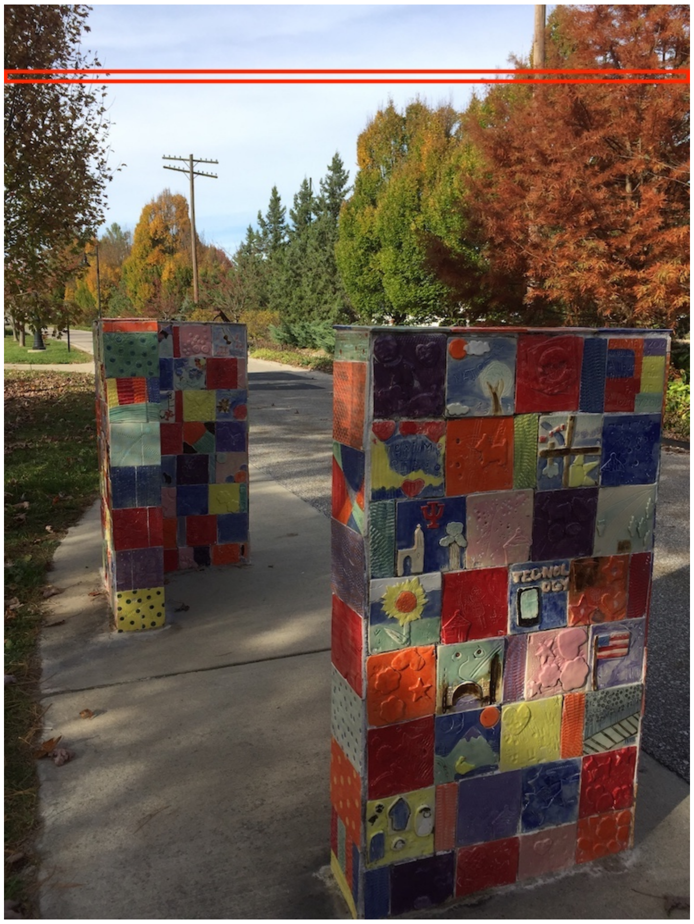

Figure 4: B-Line trail