Machine Learning for Signal Processing
(ENGR-E 511; CSCI-B 590)
Homework 1

Due date: Feb. 2, 2024, 23:59 PM (US Eastern)

Instructions

- Submission format: (Jupyter Notebook Or MatlabLive Script) + 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: MLE and MAP [3 points]

1. Prof. M loves driving, but he does not like traffic lights. In fact, he considers it is a great *success* if a traffic light is green when he arrives at it.
2. He noticed that there are 10 traffic lights on his short drive to the office from home. Once Prof. M decided to log the number of traffic lights that were green on his way to see how much luck he has in evading them.
3. Here are the logged data for 35 of his rides.
   $\big[1\ 3\ 2\ 0\ 2\ 4\ 3\ 2\ 3\ 4\ 0\ 1\ 1\ 3\ 3\ 2\ 1\ 1\ 4\ 3\ 1\ 1\ 1\ 1\ 1\ 2\ 5\ 2\ 0\ 1\ 3\ 0\ 4\ 2\ 1\ \big]$
4. Since a traffic light being green on arrival is relatively a rare incident, you can assume this process follows a Poisson distribution. Calculate the maximum likelihood estimation (MLE) to find the Poisson parameter of the underlying distribution that maximizes the likelihood of observing this data (No need to do the derivation).
5. Plot the likelihood Poisson distribution characterized by this parameter. This distribution represents the probability of $x = k$ number of lights being green on arrival given Prof. M's observations. You can use the following function to model the distribution.

```
from scipy.stats import poisson
xs = np.arange(0, 30)
ys = poisson.pmf(xs, mle)
```

6. Calculate the log likelihood for this parameter (no need to do the derivation). Also, calculate the log likelihood for a few more parameters and see if your MLE in fact gives the highest log likelihood.

7. However, Prof.M's friend does not believe in this value and said Prof. M has been rather unlucky. According to her logs on 100 rides on the same route, she has got 300 "green passes".

8. Prof. M wants to embed this prior knowledge to compute a posterior probability distribution. We know Gamma [Gamma(α, β)] distribution is a conjugate prior to Poisson –our likelihood distribution. I know, you did not specifically learn the Gamma distribution in this class, but embedding this prior knowledge in this case is easy. All you need to know is the mean of a gamma distribution is $\frac{\alpha}{\beta}$. Find α and β for the prior distribution.

9. Now you can use them to find the the maximum-a-posterior mean (λ_{pos}). All we need to know is that the posterior distribution $P(\lambda|X) \sim \text{Gamma}(\sum_i x_i + \alpha, n + \beta)$. Plot the posterior distribution on the same figure.

```
from scipy.stats import gamma
xs = np.linspace(0, 30, 300)
ys = gamma.pdf(xs, posterior_mean)
```

P2: Central Limit Theorem [3 points]

1. You're taking Prof. K's MLSP course as a residential student. It turned out that you like his offline lectures better than the recorded versions, because he tells a lot of jokes when there's no camera around. So, you decided to record his offline lectures without his permission.

2. When you played the recording at home, you realized that the recording was contaminated by some annoying classmates around your smartphone. They were too talkative in the middle of the class, making Prof. K's deep and beautiful voice inaudible.

3. You know you'll learn about source separation soon in class, but you don't know how to do it yet. But, you at least want to know which recording is better than the other, because you got to know that there's another student recorded the same lecture. For example, you could think that the recording with many interfering sources will be less preferred to the ones with less interference.

4. `x1.wav` and `x2.wav` are two recordings made in the MLSP class by two devices. Even though both of them are capturing the same part of the lecture, the numbers of interfering sources are different from each other. To assure you that this was actually about MLSP, I'm also providing `s.wav` which is Prof. K's ground-truth speech. Listen to `s.wav`, `x1.wav` and `x2.wav`. Obviously, `s.wav` must be the best among the three, but can you tell which one is cleaner between `x1.wav` and `x2.wav` (i.e. with less interfering sources)?

5. You can use the Central Limit Theorem (CLT) to figure out which one is with more sources. First, we assume that the samples from a given source are from a random variable with unknown probabilistic distribution. We assume that it's not a Gaussian distribution. For

example, draw a histogram from the samples of `s.wav`. Does it look like a Gaussian? To me it's too spiky, so maybe not. Although you cannot draw histograms of the other interfering sources (because you don't know them), let's assume they are from a non-Gaussian distribution, too.

6. According to CLT, the sum of random variables gets closer to a Gaussian distribution. If there are more random variables (i.e. sources) added up, the mixture of them will be more Gaussian-like. Therefore, you can measure the Gaussian-likeness of the sample distributions of `x1.wav` and `x2.wav` to see which one is with more sources.

7. You will use a non-Gausianity metric, called "Kurtosis," for this. Long story short, kurtosis is defined as follows if the distribution of the random variable x is normalized (i.e. zero mean and unit variance):
$$\mathcal{K}(x) = E(x^4) - 3 \tag{1}$$

8. For Python, it's convenient to import LIBROSA[1], which has a load function as follows:

```
import librosa
x, sr = librosa.load('x1.wav', sr=None)
```

(Note: the `sr=None` argument makes sure that the load function doesn't change the original sampling rate. For now don't think too much about this, and just use this argument.)

9. Standardize both signals by subtracting their sample means and by dividing by their sample standard deviation.

10. Calculate the kurtosis. Which one is less Gaussian-like according to the kurtosis values? Draw histograms of the two signals. Can you eyeball the graphs to see which one is less Gaussian-like?

11. As a sanity check, compare them with the histogram of `s.wav` as well. Can you see the mixture signals' histograms are more Gaussian-like than the ground-truth source'? Calculate the kurtosis of `s.wav` as well for a comparison.

12. Submit your histograms, kurtosis values of the three signals, and your verbal explanations about your decision as to which one is with more interfering sources. Don't forget to submit your code.

P3: Eigenvectors for Two-Notes [5 points]

1. Write your own power iteration routine that calculates the eigenvector one by one (M01-L02-S11).

2. Load `flute.mat`. It is a matrix representation of the two musical notes played by a flute. Plot this matrix by using a color map of your choice to show the intensity of all the horizontal bars (they're something called harmonics). Your plot will look like the one in M01-C02-S07.

[1] http://librosa.github.io

3. The input matrix $\boldsymbol{X}$ has 143 column vectors, each of which has 128 frequency elements. Estimate two eigenvectors from this by using the power iteration routine you wrote. Plot your eigenvectors and put them on your report along with the $\boldsymbol{X}$ matrix. This time, since the eigenvectors are multidimensional, you need to draw a graph of each of them rather than a line. Once again, it will look like the tall two-column matrix in the middle of M01-C02-S07.

4. Now you know the representative spectra for the two notes. How would you recover their temporal activations? They will be two row vectors for the two notes, respectively. You need to come up with an equation for this, and plot the activation (row) vectors you got from this procedure in the report. Once again, it will look like the third fat matrix in M01-C02-S07.

5. Finally, since you know the basis vectors and their activations, you can recover each source separately. How would you recover the first note? Come up with an equation for this, and plot the result.

6. You don't have to answer this question, but think about whether you like this separation result or not. I will teach you some better ways to do this in the future.

P4: BFGS [5 points]

1. Load `sg_train.jpg`. This is the clean image you use as the target of the optimization. Load `sgx_train.jpg`, too. It's the noisy image you use as the input training data. Your test input is `sgx_test.jpg`. Replicate the denoising procedure (including testing) from page 7 to 11 in "Lecture 03 - Optimization." Show me the denoised images (of both training and testing image).

2. On top of the gradient descent algorithm, now do the denoising using the BFGS algorithm to do the same denoising job. You know that you have to implement your own BFGS routine. Report the results.

3. Compare the convergence behavior of the two approaches in graphs.

   Hint on converting the image into the $\boldsymbol{X}$ matrix: the $\boldsymbol{f}$ vector is a vectorized version of the 2D denoising filter you're estimating (you can construct it by concatenating all the column vectors and coming up with a very long column vector). For example, $\boldsymbol{f} \in \mathbb{R}^{225\times 1}$ if your denoising filter is a 15×15 matrix. That's the filter you have to estimate. You can convert your noisy input image into $\boldsymbol{X}$ by vectorizing each patch and concatenating the column vectors horizontally, where a patch is a small section of your image with the same number of dimensions with your filter (15×15). You collect these patches from the top-left corner to the bottom-right corner of your input image by shifting it by one pixel each time. For example, for an input image of 200×200 pixels, and a 15×15 filter, you have $186 \times 186 = 34,596$ patches, each of which has 15×15 pixels. Eventually, $\boldsymbol{f}^\top \boldsymbol{X}$ will result in a cleaned-up image in the form of a row vector, which you can reshape back to the 2D image.

   Hint on logistic functions:

$$g(x) = \frac{1}{1 + \exp(-x)} \tag{2}$$

$$g'(x) = g(x)\big(1 - g(x)\big) \tag{3}$$