Machine Learning for Signal Processing (ENGR-E 511) Homework 4

Due date: April 12, 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: When to applaud? [4 points]

1. `Piano_Clap.wav` is an audio signal simulating a music concert, particularly at the end of a song. As some of the audience haven't heard of the song before, they started to applaud before the end of the song (at around 1.5 seconds). Check out the audio.
2. Since I'm so kind, I did the MFCC conversion for you, which you can find in `mfcc.mat`. If you load it, you'll see a matrix `X`, which holds 962 MFCC vectors, each of which has 12 coefficients.
3. You can find the mean vectors and covariance matrices in `MuSigma.mat` that I kindly provide, too. Once you load it, you'll see the matrix `mX`, which has two column vectors as the mean vectors. The first column vector of `mX` is a 12-dimensional mean vector of MFCCs of the piano-only frames. The second vector holds another 12-dimensional mean vector of MFCCs of the claps. In addition to that, `Sigma`, has a 3D array (12×12×2), whose first 2D slice (12×12) is the covariance matrix of the piano part, and the upper 2D slice is for the clap sound.
4. Since you have all the parameters you need, you can calculate the p.d.f. of an MFCC vector in `X` for the two multivariate normal (Gaussian) distribution you can define from the two sets of means and cov matrices. Go ahead and calculate them. Put them in a 2×962 matrix:

$$\boldsymbol{P} = \begin{bmatrix} P(\boldsymbol{X}_1|\mathcal{C}_1) & P(\boldsymbol{X}_2|\mathcal{C}_1) & \cdots & P(\boldsymbol{X}_{962}|\mathcal{C}_1) \\ P(\boldsymbol{X}_1|\mathcal{C}_2) & P(\boldsymbol{X}_2|\mathcal{C}_2) & \cdots & P(\boldsymbol{X}_{962}|\mathcal{C}_2) \end{bmatrix}. \tag{1}$$

Note that $\mathcal{C}_1$ is for the piano frames while $\mathcal{C}_2$ is for the applause. Normalize this matrix, so that you can recover the posterior probabilities of belonging to the two classes given an MFCC frame:

$$\tilde{\boldsymbol{P}}_{c,t} = \frac{P(\boldsymbol{X}_t|\mathcal{C}_c)}{\sum_{c=1}^{2} P(\boldsymbol{X}_t|\mathcal{C}_c)} \tag{2}$$

Plot this 2×962 matrix as an image (e.g. using the `imagesc` function in MATLAB). This is your detection result. If you see a frame at t' where $\tilde{\boldsymbol{P}}_{1,t'} < \tilde{\boldsymbol{P}}_{2,t'}$, it could be the right moment to start clapping.

5. You might not like the this result, because it is sensitive to the wrong claps in the middle. You want to smooth them out. For this, you may want to come up with a transition matrix with some dominant diagonal elements:

$$\boldsymbol{T} = \begin{bmatrix} 0.9 & 0.1 \\ 0 & 1 \end{bmatrix}. \tag{3}$$

What it means is that if you see a $\mathcal{C}_1$ frame, you'll want to stay at that status (no clap) in the next frame with a probability 0.9, while you want to transit to $\mathcal{C}_2$ (clap) with a probability of 0.1. On the other hand, you absolutely want to stay at $\mathcal{C}_2$ once you observe a frame with that label (you don't want to get back if you start clapping).

Apply this matrix to your $\tilde{\boldsymbol{P}}$ matrix in a recursive way:

$$\bar{\boldsymbol{P}}_{:,1} = \tilde{\boldsymbol{P}}_{:,1} \tag{4}$$
$$b = \arg\max_c \bar{\boldsymbol{P}}_{c,t} \tag{5}$$
$$\bar{\boldsymbol{P}}_{:,t+1} = \boldsymbol{T}^{\top}_{b,:} \odot \tilde{\boldsymbol{P}}_{:,t+1} \tag{6}$$
$$\tag{7}$$

First, you initialize the first column vector of your new posterior prob matrix $\bar{\boldsymbol{P}}$ (you have no previous frame to work with at that moment). For a given time frame of interest, $t+1$, you first need to see its previous frame to find which class the frame belongs to (by using the simple max operation on the posterior probabilities at t). This class index b is going to be used to pick up the corresponding transition probabilities (one of the row vectors of the $\boldsymbol{T}$ matrix). Then, the transition probabilities will be multiplied to your existing posterior probabilities at $t+1$.

In the end, you may want to normalize them so that they can serve as the posterior probabilities:

$$\begin{aligned} \bar{\boldsymbol{P}}_{1,t+1} &= \bar{\boldsymbol{P}}_{1,t+1} / \left(\bar{\boldsymbol{P}}_{1,t+1} + \bar{\boldsymbol{P}}_{2,t+1}\right) \\ \bar{\boldsymbol{P}}_{2,t+1} &= \bar{\boldsymbol{P}}_{2,t+1} / \left(\bar{\boldsymbol{P}}_{1,t+1} + \bar{\boldsymbol{P}}_{2,t+1}\right). \end{aligned} \tag{8}$$

Repeat this procedure for all the 962 frames.

6. Plot your new smoothed post prob matrix $\bar{\boldsymbol{P}}$. Do you like it?

7. If you don't like the naïve smoothing method, there is another option, called the Viterbi algorithm. This time, you'll calculate your smoothed post prob matrix in this way:

$$\bar{\boldsymbol{P}}_{:,1} = \tilde{\boldsymbol{P}}_{:,1} \tag{9}$$

First, initialize the first frame post prob as usual. Then, for $(t+1)$-th frame, we will construct $\bar{\boldsymbol{P}}_{c,t}$. For this, we need to see t-th frame, but this time we check on all paths to pick up the best one by calculating all the probabilities of state transitions from c at t to c' at $t+1$. For example, from (c,t) to $(c'=1,t+1)$:

$$b = \arg\max_{c} \boldsymbol{T}_{c,1}\bar{\boldsymbol{P}}_{c,t}. \tag{10}$$

This will tell you which of the two previous states $c=1$ and $c=2$ at t was the best transition to the current state $\mathcal{C}_1$ at $t+1$ (we're seeing only $\mathcal{C}_1$ for now). If $b=1$, it's more probable that the previous state at t was $\mathcal{C}_1$ rather than $\mathcal{C}_2$. We keep a record in our $\boldsymbol{B}$ matrix for this best choices:

$$\boldsymbol{B}_{1,t+1} = b \tag{11}$$

Eventually, we use this best transition probability to construct the post prob of $\mathcal{C}_1$ at $t+1$:

$$\bar{\boldsymbol{P}}_{1,t+1} = \boldsymbol{T}_{b,1}\bar{\boldsymbol{P}}_{b,t}\boldsymbol{P}_{1,t+1} \tag{12}$$

$$\tag{13}$$

In other words, the prior probability (the transition probability and the accumulated post prob in the t-th frame), $\boldsymbol{T}_{b,1}\bar{\boldsymbol{P}}_{b,t}$, is multiplied to the likelihood $\boldsymbol{P}_{1,t+1}$ to form the new post prob.

We keep this best previous state sequences. For example, $\boldsymbol{B}_{1,102}$ will hold the more likely previous state at 102-th frame if that frame's state is $\mathcal{C}_1$, while $\boldsymbol{B}_{2,102}$ records its corresponding 101-th state that could have transit to $\mathcal{C}_2$.

8. Don't forget to repeat this for $\bar{\boldsymbol{P}}_{2,t+1}$.

9. Don't forget to normalize $\bar{\boldsymbol{P}}$ as in (8).

10. Once you construct your new post prob matrix from the first frame to the 962-th frame, you can back track. Choose the larger one from $\bar{\boldsymbol{P}}_{1,962}$ and $\bar{\boldsymbol{P}}_{2,962}$. Let's say $\bar{\boldsymbol{P}}_{2,962}$ is larger. Then, refer to $\boldsymbol{B}_{2,962}$ to decide the state of the 961-th frame, and so on.

11. Report this series of states as your backtracking result (e.g. as a plot). This will be the hidden state sequence you wanted to know. Is the start of the applause making more sense to you?

P2: Multidimensional Scaling [3 points]

1. `MDS_pdist.mat` holds a 996×996 square matrix, which holds squared pairwise Euclidean distances that I constructed from a set of data points. Of course I won't share the original data samples with you. Instead, you'll write a code for multidimensional scaling so that you can recover the original map out of $\boldsymbol{L}$. If your MDS routine is successful, you should be able to see what the original map was. The original map was in the 2D space, so you may want to see two largest eigenvectors. Report the map you recovered in the form of a scatter plot (dots represent the location of the samples).

2. Note that the MDS result can be rotated and shifted.

P3: Kernel PCA [4 points]

1. `concetric.mat` contains 152 data points each of which is a 2-dimensional column vector. If you scatter plot them on a 2D space, you'll see that they form the concentric circles you saw in class.

2. Do kernel PCA on this data set to transform them into a new 3-dimensional feature space, i.e. $\boldsymbol{X} \in \mathbb{R}^{2\times 152} \Rightarrow$ Kernel PCA $\Rightarrow \boldsymbol{Z} \in \mathbb{R}^{3\times 152}$.

3. In theory, you have to do the centering and normalization procedures in the feature space, but for this particular data set, you can forget about it.

4. You remember how the after-transformation 3D features look like, right? Now your data set is linearly separable. Train a perceptron that does this linear classification. In other words, your perceptron will take the transformed version of your data (a 3-dimensional vector) and do a linear combination with three weights $\boldsymbol{w} = [w_1, w_2, w_3]^\top$ plus the bias term b:

$$y_i = \sigma\left(\boldsymbol{w}^\top \boldsymbol{Z}_{:,i} + b\right), \tag{14}$$

where y_t is the scalar output of your perceptron, σ is the activation function you define. If you choose logistic function as your activation, then you'd better label your samples with 0 or 1. You know, the first 51 samples are for the inner circle, so their label will be 0, while for the other outer ring samples, I'd label them with 1.

You'll need to write a backpropagation algorithm to estimate your parameters $\boldsymbol{w}$ and b, which minimize the error between y_i and t_i. There are a lot of choices, but you can use the sum of the squared error: $\sum_i \frac{1}{2}(y_i - t_i)^2$. Note that this dummy perceptron doesn't have a hidden layer. Note that you may want to initialize your parameters with some small (uniform or Gaussian) random numbers centered around zero.

5. Your training procedure will be sensitive to the choice of the learning rate and the initialization scheme (the range of your random numbers). So, you may want to try out a few different choices. Good luck!

P4: Neural Networks [4 points]

1. Build a neural network that has a single hidden layer for the concentric circles problem. Instead of taking the kernel PCA results as the input, your neural network will take the $\boldsymbol{X}$ matrix as the input. Therefore, instead of a perceptron with 3 input units, now your neural network will have 2 input units, each of which will take one of the coordinates of a sample.

2. As we skip kernel PCA, we will convert this 2D data point into a 3D feature vector, which will correspond to the first layer of your neural network:

$$\boldsymbol{X}^{(2)}_{:,i} = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{X}^{(1)}_{:,i} + \boldsymbol{b}^{(1)}\right), \tag{15}$$

where $\boldsymbol{X}^{(1)}_{:,i} \in \mathbb{R}^{2\times 1}$ is the input to the network (i-th column vector of $\boldsymbol{X}$), while $\boldsymbol{X}^{(2)}_{:,i} \in \mathbb{R}^{3\times 1}$ is the output of the hidden units. What that means is that the weight matrix $\boldsymbol{W}^{(1)} \in \mathbb{R}^{3\times 2}$ and the bias vector $\boldsymbol{b}^{(1)} \in \mathbb{R}^{3\times 1}$.

3. In the final layer, you'll do the usual linear classification on $\boldsymbol{X}_{:,i}^{(2)}$, which will be similar to the perceptron you developed in Problem 3, although this time the input to the perceptron is $\boldsymbol{X}_{:,i}^{(2)}$, not the kernel PCA results:

$$y_i = \sigma\left(\left(\boldsymbol{w}^{(2)}\right)^\top \boldsymbol{X}_{:,i}^{(2)} + b^{(2)}\right) \tag{16}$$

4. Note that you train these two layers altogether. Your backpropagation should work from the final layer back to the first layer. Eventually, you want to estimate these parameters with your backpropagation: $\boldsymbol{W}^{(1)} \in \mathbb{R}^{3\times 2}, \boldsymbol{w}^{(2)} \in \mathbb{R}^{3\times 1}, \boldsymbol{b}^{(1)} \in \mathbb{R}^{3\times 1}, b^{(2)} \in \mathbb{R}^1$.

5. Do not use any of the deep learning frameworks, such as Tensorflow, PyTorch or MATLAB's neural network toolboxes, to get around the differentiation. Write up your own backpropagation routine and update algorithms.