Machine Learning for Signal Processing
(ENGR-E 511)
Homework 6

Due date: Apr. 24, 2022, 23:59 PM (US Eastern)

Instructions

- Submission format: Jupyter 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.
  - Google Colab is the best place to begin with if this is the first time using iPython notebook.
  - 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: Stereo Matching (revisited) [5 points]

1. `im0.ppm` (left) and `im8.ppm` (right) are the pictures taken by two different camera positions[1]. If you load the images, they will be a three dimensional array of $381 \times 430 \times 3$, whose third dimension is for the three color channels (RGB). Let's call them X^L and X^R. For the (i,j)-th pixel in the right image, $X^R_{(i,j,:)}$, which is a 3-d vector of RGB intensities, we can scan and find the most similar pixel in the left image at i-th row (using a metric of your choice). For example, I did the search from $X^L_{(i,j,:)}$ to $X^L_{(i,j+39,:)}$, to see which pixel among the 40 are the closest. I record the index-distance of the closest pixel. Let's say that $X^L_{(i,j+19,:)}$ is the most similar one to $X^R_{(i,j,:)}$. Then, the index-distance is 19. I record this index-distance (to the closest pixel in the left image) for all pixels in my right image to create a matrix called "disparity map", $\boldsymbol{D}$, whose (i,j)-th element says the index-distance between the (i,j)-th pixel of the right image and its closest pixel in the left image. For an object in the right image, if its pixels are associated with an object in the left image, but are shifted far away, that means the object is close to the camera, and vice versa.

2. Calculate the disparity map $\boldsymbol{D}$ from `im0.ppm` and `im8.ppm`, which will be a matrix of 381×390 (since we search within only 40 pixels). Vectorize the disparity matrix and draw a histogram. How many clusters do you see?

3. Write up your own GMM clustering code, and cluster the disparity values in $\boldsymbol{D}$. Each value will belong to (only) one of the clusters. The number of clusters says the number of depth

[1] http://vision.middlebury.edu/stereo/data/

levels. If you replace the disparity values with the cluster means, you can recover the depth map with k levels. Plot your depth map (the disparity map replaced by the mean disparities as in the image quantization examples) in gray scale–pixels of the frontal objects should be bright, while the ones in the back get darker.

4. Extend your implementation with the MRF's smoothing priors using an eight neighborhood system (e.g. $\mathcal{N}_{i,j} = \big\{(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)\big\}$. Feel free to choose either ICM or Gibbs sampling. Show me the smoothed results. You can use the Gaussian-kernel-looking prior probability equations discussed in class (M10 S8).

P2: Probabilistic Latent Semantic Indexing (PLSI) for Speech Denoising [5 points]

1. Convert the two training signals `trs.wav` and `trn.wav` using the earlier STFT setup you used in the previous homework. Let's call these complex-valued matrices $\boldsymbol{S}$ and $\boldsymbol{N}$.

2. Build a speech denoising system by using the PLSI algorithm. The overall procedure is described in M11-S33. The update rules are on M11-S18 (use the ones on the top of the slide, not the NMF ones).

3. First, run your PLSI algorithm to learn $\boldsymbol{B}^{(1)}$ from $|\boldsymbol{S}|$ (the magnitudes).

4. Second, run your PLSI algorithm to learn $\boldsymbol{B}^{(2)}$ from $|\boldsymbol{N}|$.

5. Third, load your test mixture `tex.wav` and turn it into a spectrogram $\boldsymbol{X}$. Run your third PLSI routine here to learn $\boldsymbol{\Theta}^{(3)}$ by taking the magnitudes $|\boldsymbol{X}|$. Remember, you don't want to update $\boldsymbol{B}^{(1)}$ and $\boldsymbol{B}^{(2)}$ during testing. You just initialize them from the ones you trained.

6. You know, the speech source can be first recovered by doing $\boldsymbol{B}^{(1)}\boldsymbol{\Theta}^{(3)}_{1:K_s,:}$, where K_s is the number of basis vectors in $\boldsymbol{B}^{(1)}$. It means that you need to pick your own choice of K_s and K_n during training.

7. However, $\boldsymbol{B}^{(1)}\boldsymbol{\Theta}^{(3)}_{1:K_s,:}$ will only give you the "probability matrix" of the speech source, not the actual spectrogram. Hence, you need to turn it into some kind of TF-bin-wise posterior probability (of belonging to the speech source) and use it as a mask:

$$\hat{\boldsymbol{S}}_{\text{test}} = \frac{\boldsymbol{B}^{(1)}\boldsymbol{\Theta}^{(3)}_{1:K_s,:}}{[\boldsymbol{B}^{(1)},\boldsymbol{B}^{(2)}]\boldsymbol{\Theta}^{(3)}} \odot \boldsymbol{X}, \tag{1}$$

   where $[\boldsymbol{B}^{(1)},\boldsymbol{B}^{(2)}]$ must be a $F \times (K_s + K_n)$ matrix and $\odot$ is a Hadamard product.

8. Transform $\hat{\boldsymbol{S}}_{\text{test}}$ back to the time domain.

9. Report the SNR value of the separation result by comparing $\hat{\boldsymbol{s}}_{\text{test}}$ to `tes.wav`.

10. Plug in the recovered speech to the notebook with a player (so that the AI can listen to it).

P3: PLSI for Analyzing Twitter Stream [5 points]

1. `twitter.mat` holds two Term-Frequency (TF) matrices $\boldsymbol{X}_{tr}$ and $\boldsymbol{X}_{te}$. It also contains $Y_{tr}Mat$ and $Y_{te}Mat$, the target variables in the one-hot vector format.

2. Each column of the TF matrix $\boldsymbol{X}_{tr}$ can be either "positive", "negative", or "neutral", which are represented numerically as 1, 2, and 3 in the $Y_{tr}Mat$. They are sentimental classes of the original twits.

3. Learn 50 PLSI topics $\boldsymbol{B} \in \mathbb{R}^{891\times 50}$ and their weights $\boldsymbol{\Theta}_{tr} \in \mathbb{R}^{50\times 773}$ from the training data $\boldsymbol{X}_{tr}$, using the ordinary PLSI update rules.

4. Reduce the dimension of $\boldsymbol{X}_{te}$ down to 50, by learning the weight matrix $\boldsymbol{\Theta}_{te} \in \mathbb{R}^{50\times 193}$. This can be done by doing another PLSI on the test data $\boldsymbol{X}_{te}$, but this time by reusing the topic matrix $\boldsymbol{B}$ you learned from the training set. So, you skip the update rule for $\boldsymbol{B}$. You only update $\boldsymbol{\Theta}_{te} \in \mathbb{R}^{50\times 193}$.

5. Define a perceptron layer for the softmax classification. This part is similar to the case with kernel PCA with a perceptron as you did in Homework #5. Instead of the kernel PCA results as the input to the perceptron, you use $\boldsymbol{\Theta}_{tr}$ for training, and $\boldsymbol{\Theta}_{te}$ for testing. This time the number of output units is 3 as there are three classes, and that's why the target variable $Y_{tr}Mat$ is with three elements. Review M6 S37-39 to review what softmax is.

6. Report your classification accuracy.

7. Note: do NOT use any deep learning framework that supports softmax implementation and automatic gradient computation. Use your own implementation of softmax and backpropagation.