CS598 PS 2020 – PROBLEM SET 2

This problem set is due on October 2nd. Scanned/handwritten PDFs will not be accepted. Try to work on these problems by yourself, the code you will write in this homework will make your life easier down the road. Code submission should be as a Python notebook, we will not accept solutions that use third party code for PCA, ICA, NMF and ISOMAP., please implement these functions yourself.

Problem 1. An audio features project

Download the sound-file `vl1.wav`. This is the sound of a 80s synthesizer that used to be very hip. The drumbeat in this sound is composed out of three "instruments". Each one is distinct from the others in its spectral composition and you should be able to tell them apart by listening to it. Your computer however cannot (yet).

Obtain the magnitude of the spectrogram of this recording. Use a window size of 1024 points, an overlap ¾ of the window size and apply a Hamming window. To make the spectrogram easy to see as an image plot, display the square root of its magnitude. By looking at it you should be able to "see" the difference between the three instruments.

Let's make your computer discover the three instruments in the beat:

1. Perform PCA on the magnitude spectrogram and extract three components. The spectrum (frequency) axis will be the input dimensions axis, and each time slice of the spectrogram will be a sample. Your PCA components should be three "eigenspectra" with as many elements as frequencies.
2. Then extract three ICA components. Remember to do PCA beforehand, you cannot drop the dimensionality with ICA only.
3. Then extract three NMF components (no need for PCA this time).

Plot the resulting features and weights for each of the three above cases. Note that in the case of PCA these will be the eigenvectors of the covariance. In ICA they will be the columns of what is known as the *mixing matrix* (when you perform ICA using $\mathbf{y} = \mathbf{W}\mathbf{x}$, the mixing matrix is the inverse of $\mathbf{W}$). In NMF they will be the columns of the $\mathbf{W}$ matrix, assuming the decomposition spectrogram $= \mathbf{W}\mathbf{H}$. What observations can you make? How do the results differ? Which ones make the most sense? Why?

Problem 2. Handwritten digit features

Download the data file `digits-labels.npz`. Load the data from that file and look for variable `d`. It contains a handwritten digit collection of 28×28 images that is used for handwriting recognition benchmarks. The pixel data will be ordered as a matrix so that each column is an unwrapped digit image (therefore of $28 \times 28 = 784$ dimensions). To display the ith data points you can do: `imshow(reshape( d[:,i],(28,28),'F'))`. Do so and make sure that the data is loaded correctly and does indeed contain digits.

Just as before, perform PCA, ICA and NMF. Drop the dimensionality to 36 dimensions for all cases. Plot the resulting features from each transform as images (use the one-liner above for the eigenvectors, mixing matrix columns, etc). Make observations on how and why they differ.

Problem 3. The geometry of handwritten digits

In the `digits-labels.npz` file there is also another variable, vector `l`, that contains the digit label corresponding to each column of the matrix `d`.

Select only the columns that correspond to the digit 6. Perform PCA and drop the dimensionality down to two dimensions. Instead of plotting the features like we did above, plot the projection as a 2D scatter plot. Instead of dots display a small image of the digit corresponding to every point in the scatter plot (if it's unreadable use only a subset of the digits to make a cleaner plot). Make observations on how the shape of the shown digits changes as it is distributed across the 2D space. Redo this experiment using the embedding from ISOMAP. For the computation of the distance matrix/graph use only the 6 nearest neighbors. Plot the resulting embedding as before. What's different now?

If you have nothing better to do and you have an urge to impress me, try the same thing with other manifold algorithms and show/compare the results.

Enjoy!

Some implementation notes:

- For the ICA algorithm you have a lot of choices, I suggest you implement the algorithm in slide 17 of the ICA lecture. This is known as the infomax algorithm if you need to find more info inline. Usually, we use $f() = 2\tanh()$. Note that the algorithm as shown in the slide is an update based on a single input vector. You can loop through that, or you can use a batch operation. If you do that the identity matrix needs to be multiplied with the number of samples in your batch, i.e. for $\mathbf{y} \in \mathbb{R}^{M \times N}, \Delta\mathbf{W} \propto \left(N\mathbf{I} - 2\tanh\left(\mathbf{y}\right)\cdot\mathbf{y}^\top\right)\cdot\mathbf{W}$. This should converge in all problems in about 500 iterations. Just to be safe run it a couple of times when you use it and see what comes out. Iterative algorithms are flaky and sometimes do not converge. If it repeatedly doesn't work then you have a problem though.

- When implementing NMF there are a bunch of divisions in the update algorithms. Get in the habit of adding a small constant to the denominator (e.g. 1e-6) to avoid divisions by zero. NMF should converge in 100 iterations or so for all problems in this homework (and elsewhere!).
- ISOMAP is a little more tricky to implement. Start by implementing MDS, make sure it works as expected, and then incorporate geodesic distances. You can use Floyd's algorithm to get these from the the neighbor graph.
- When plotting the digits in a 2d plane, you can use a subset of the available data so that you do not get an overly busy plot. You can also set white to be transparent so that you do not hide background digits.
- When loading sounds in Python make sure that the data is converted to a floating point format. Many functions load them as integers which then makes any subsequent operations very inaccurate.
- There are many ways to plot the necessary quantities, experiment and see what works best. Don't ask us how to do it, this is an important skill to develop on your own. In fact, the problems sets in this class are intentionally not very specific so that you can start exercising your judgement and taking initiative (as you will have to do in real-life research).
- Excluding gratuitous commenting, the PCA, ICA, NMF and ISOMAP functions in Python should not take more than 15 or so lines of code. All these problems should comfortably run on a 10-year-old laptop.
- I understand that you might love MATLAB, or Fortran, or whichever other language, but we can only afford to deal with one language in this class. Given that Python has become the standard language for both ML and SP, we'll stick to that.