Eigen Math & Frequency Analysis for Audio

December 3, 2017

Learning Goals

- Understand the difference between correlation and covariance matrix and how they are applied in different situations.
- Understand what singular vector decomposition yields and learn how to implement SVD in MATLAB.
- Apply the knowledge from facial recognition and implement for voice classification via MATLAB.
- Be able to conduct processing audio data in MATLAB.
- Understand the complex math concept from external sources, such as academic paper.

Eigen Analysis Revisited (3 hours)

What did we learn?

During QEA I Module II, we learned how to recognize faces by conducting Principal Component Analysis, also known as Eigenfaces method. Basic idea for voice recognition works in similar fashion: a voice can be classified based on its variation from the entire mean of the voice data. Since **variance** can be also interpreted as the distance, or difference from the mean of the data, we can apply the same math from Module II for recognizing voices, such as Singular Vector Decomposition. Since it has been to long from middle of the first semester, we will review mathematical concept.

Principal Component Analysis - Correlation and Covariance

Principal Component Analysis (PCA) uses an orthogonal transformation to convert a data set to a set of values of linearly uncorrelated variables called principal components.The number of distinct principal components is equal to the smaller of the number of original variables or the number of observations minus one (N-1). The transformation is defined that the first principal component has the largest possible variance, second principal component has the second largest possible variance and so on. Since the components are orthogonal to the preceding components, the resulting vectors are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables. Conducting PCA can be done by eigenvalue decomposition of a data covariance or correlation matrix or singular value decomposition of a data matrix, usually after mean centering (and normalizing) the data matrix for each elements. (Referenced)

A correlation matrix is a matrix showing correlation coefficients between sets of variables. Higher correlation coefficients mean two different data tips are more related to each other. Therefore, if a data matrix is defined as following:

$$\textbf{Data} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

The correlation matrix is:

$$\textbf{Correlation Matrix(A)} = \frac{1}{\sqrt{N}} \begin{bmatrix} \frac{x_1-\mu_x}{\sigma_x} & \frac{y_1-\mu_y}{\sigma_y} \\ \frac{x_2-\mu_x}{\sigma_x} & \frac{y2-\mu_y}{\sigma_y} \\ \frac{x_3-\mu_x}{\sigma_x} & \frac{y_3-\mu_y}{\sigma_y} \\ \cdots & \cdots \end{bmatrix}$$

Where N is the number of rows in the matrix, μ_x, μ_y are the mean of the data set, and σ_x, σ_y are the standard deviation of the each column.

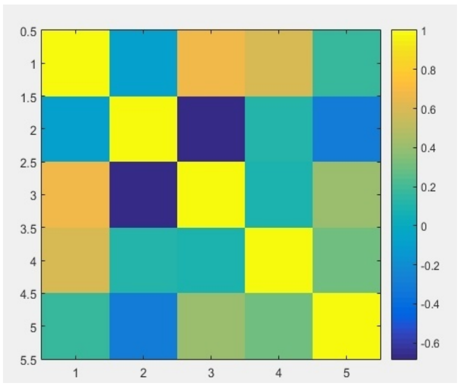

Figure 1: Example of Correlation Matrix Plot

(Referenced)

A covariance matrix is a matrix built of covariance coefficients, which are the measures of how changes in one variable are associated with changes in a second variable. Essentially, covariance coefficient indicates the degree of linear association between two variables. Therefore, covariance matrix can be defined as

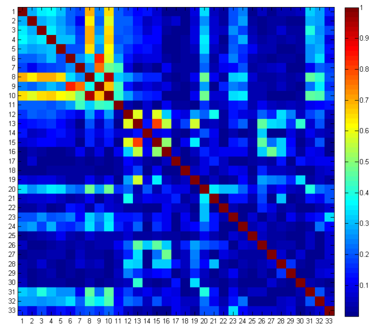

Figure 2: Example of Covariance Matrix Plot

(Referenced)

$$\textbf{Covariance Matrix(B)} = \frac{1}{N-1} \begin{bmatrix} (x_1-\mu_1)(x_1-\mu1) & (y_1-\mu_1)(y_1-\mu1) \\ (x_2-\mu_2)(x_1-\mu1) & (y_2-\mu_2)(x_2-\mu2) \\ \cdots & \cdots \\ (x_n-\mu_n)(x_1-\mu1) & (y_n-\mu_n)(y_1-\mu2) \end{bmatrix}$$

Then, how should we choose what matrix to use for Principal Component Analysis? Covariance matrix is commonly used when variables are on the same scale. If the variables are on different scale, correlation matrix is used. For example, if the variables have different units of measure, such as pounds and inches. Since you cannot compare pounds and inches directly, you should use correlation matrix. If you want to conduct PCA on the temperature data for different cities, since the units of measure are the same (Either Fahrenheit or Celsius), covariance matrix should be used. In cases where scales are different, covariance matrix can also adjust the scales of the variables.

For example, if you want to conduct PCA on population for different states, using covariance matrix would account for the size of each states.

Exercise 1

Load sample data in MATLAB by typing *load hospital*. Then, with *[X = hospital.Weight, hospital.BloodPressure]*, plot the correlation and covariance matrix of the data and find how weight and blood pressure are related. Also, state how correlation and covariance matrix are different.

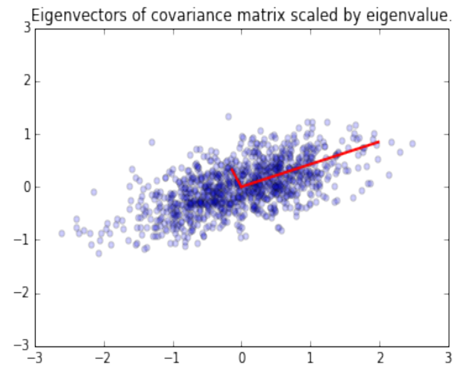

Figure 3: Example of PCA (Referenced)

Principal Component Analysis - Singular Value Decomposition

Helpful Resource: CMU CS Theory

Singular Value decomposition is decomposition of a matrix M (m by n) whose entries come from the field K, which is either the field of real numbers or the field of complex numbers. The decomposition takes the form of $\mathbf{M} = \mathbf{U\Sigma V}^*$, where **U** is m by m unitary matrix (if K = ℝℝ , unitary matrices are orthogonal matrices), $\mathbf{\Sigma}$ is a diagonal m by n matrix with non-negative real numbers on the diagonal, **V** is an n by n unitary matrix over K, and $\mathbf{V}^*$ is the conjugate transpose of V.

The diagonal entries σ_i of $\mathbf{\Sigma}$ are known as the singular values of M. A common convention is to list the singular values in descending order. In the other words, the first entry of the singular value (eigenvalue) represents how much of eigenvector corresponding to the eigenvalue is represented in the data. The second entry of the singular value represents the second most eminent direction how data is projected and so on. Therefore, by picking certain number of singular value, you can scale your data set and this process is called Principal Component Analysis. (Referenced)

Exerise 2

Load sample data in MATLAB by typing *load hald*. Conduct SVD on one of four variables and plot the resultant data. These resources might be helpful: SVD MATLAB documentation and SVD MATLAB tutorial

Frequency Analysis and Filtering (3 hours)

Discrete Fourier Transform (DFT)

*This section reviews and builds off section of the overnight from Module 2 Night 3.

Key ideas reviewed:

- You can sample a continuous, real-world signal by sampling at discrete time steps using a simple matrix

- You can use these more simple, digital responses to analyze a signal
- You can convert a signal in the frequency domain back to the time domain

Recall that when a computer (or any digital device) samples an audio source, it samples it at a set frequency, or sampling rate. Basically a computer has a special type of clock in it that allows it to perform sampling operations only a certain number of times during a period of time.

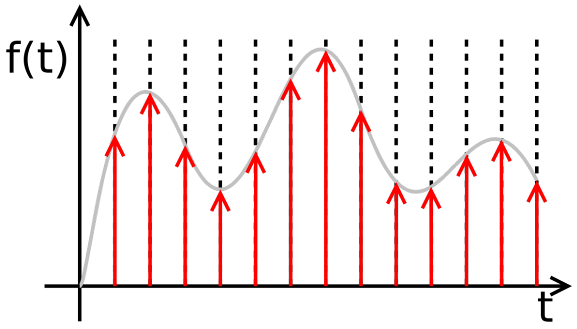

Figure 4: A continuous signal (in grey) being sampled at a specified sample rate (red arrows).

As in figure 4, a continuous sound signal can be broken up, or sampled, at different points. As you can imagine, if you don't sample at a high enough frequency, you will lose valuable information about your continuous signal. This is called aliasing. As you also might remember, as per the *Nyquist Sampling Theorem*, you only have to sample at twice the frequency of the highest frequency present in your continuous signal to be able to fully reconstruct your continuous signal.

Once you have turned a analog, continuous signal into a digital, discrete signal in the time domain, you can project it into the frequency domain which allows for rich analysis of the information within the signal.

If you'd like to watch a quick (4 minutes) refresher video on the DFT, here is a good video that explains how it works.

Recall that the Discrete Fourier Transform (DFT) works by taking a signal and sampling at a certain frequency Fs and then correlating how much of each sinusoidal basis function is in each sample.

Let's say our sampled signal looks like 1 where N is the time where the signal was sampled:

$$x = \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix} \tag{1}$$

Now we need to generate a matrix of the sinusoidal basis functions which are all just sinusoidal functions that are dependent on how many samples were taken and were they are in the sample range.

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \exp(-\frac{2\pi j}{N})2 & \exp(-\frac{2\pi j}{N})3 & \dots & \exp(-\frac{2\pi j}{N})(N-1) \\ 1 & \exp(-\frac{3\pi j}{N})2 & \exp(-\frac{2\pi j}{N})3 & \dots & \exp(-\frac{2\pi j}{N})(N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \exp(-\frac{(N-1)\pi j}{N})2 & \exp(-\frac{(N-1)\pi j}{N})3 & \dots & \exp(-\frac{(N-1)\pi j}{N})(N-1) \end{bmatrix} \tag{2}$$

This vector can then be used to calculate the weights of W on x as

$$a = Wx \tag{3}$$

Once you get these weights, you can plot them and see the frequency response of the signal in the frequency domain. This is what the *fft()* function does in Matlab.

To covert the signal back to time domain, you can use a similar matrix and the weights to get time function components.

$$W^{-1} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \exp(\frac{2\pi j}{N})2 & \exp(\frac{2\pi j}{N})3 & \dots & \exp(\frac{2\pi j}{N})(N-1) \\ 1 & \exp(\frac{3\pi j}{N})2 & \exp(\frac{2\pi j}{N})3 & \dots & \exp(\frac{2\pi j}{N})(N-1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \exp(\frac{(N-1)\pi j}{N})2 & \exp(\frac{(N-1)\pi j}{N})3 & \dots & \exp(\frac{(N-1)\pi j}{N})(N-1) \end{bmatrix} \tag{4}$$

Band Pass Filtering

Spectral Subtraction

Audio Preprocessing (2 hours)

Now that we've learned how to build two different types of filters, we're going to learn how to apply them to audio clips. As we get into that, however, we will rapidly realize the length of the word matters greatly. Since each person recording their clip didn't say their word at exactly the same time, we are going to need to correct this by trimming the clips to all start at exactly the same time and last for the same duration as can be see in figure 5.

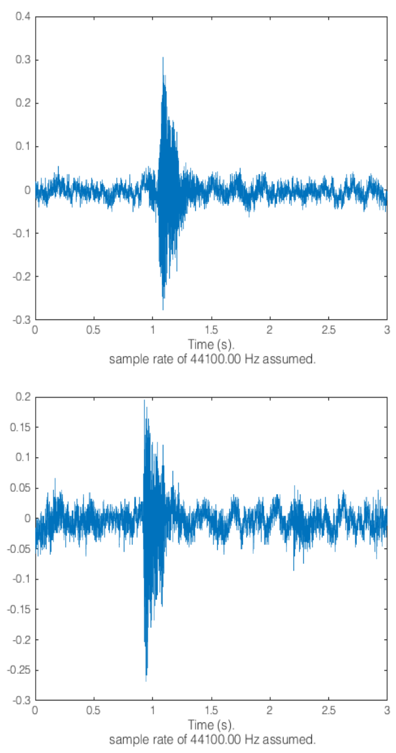

Figure 5: Two raw audio clips before processing

Algorithm Creation

To start trimming these clips to length, we should start by observing them. Obviously the levels of the word are significantly higher than other times during the clip. However, some clips have more fluctuation in their levels surrounding the words, so we'll need to apply our filtering.

After filtering is done (as can be seen in figure 6), after applying the band pass filter, our data is significantly cleaner and devoid of noise. This helps our 'word part' stand out much more.

To isolate the word consistently, we shall define a threshold and gather a section of the clip surrounding the point where this threshold is triggered. In essence, we will look through all the sample points of the signal and when the level goes above 0.1, we shall take out a sub clip of 0.5s surrounding that point.

Exerise 1

Build an algorithm to trim all the sound clips to be the same length and contain the word part. Work on this for an hour, and if you do not come up with something, feel free to use the sample script found here. We don't want you to struggle on a relatively simple algorithm, but feel it would help familiarize you with the format of the data sets.

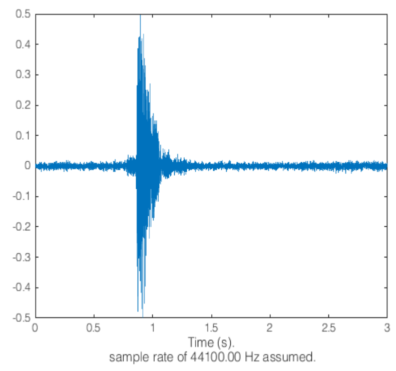

Figure 6: Two raw audio clips before processing

Finding Average Voice Going Further

Exercise 2

Just as you did in the *Eigen Analysis Revisited* section, find the average voice of the data set. (Hint: Start by putting all the sound clips in the frequency domain. Remember to use your cleaned data set that you just made!)

*Exercise 3 *Optional*

Now that you have found the average voice, feel free to try the eigen analysis section using the sound data sets and the average voice you just found.