METROPOLITAN STATE UNIVERSITY℠ OF DENVER

Digital Image Processing
Project 1: PCA Face Recognition

Steven Jennings 2 April 2018

Overview

The goal of this assignment was to detect a face in an image. This goal was to be accomplished through the following parts:

- Training the programing with a given dataset
    - Vectorizing the images
    - Calculating the mean image of the entire dataset (Common facial features)
    - Subtracting the mean image from each image in the dataset (Distinguished facial features)
    - Calculating the covariance matrix with the mean-subtracted image dataset
    - Calculating the eigenvalues, eigenvectors, and eigenfaces of the dataset
    - Calculating the weights of the dataset (for a test image to test against)
- Testing an image against the given dataset
    - Subtracting the mean image of the dataset from the test image
    - Projecting the test image into eigenspace
    - Calculating weights of the test image
- Determine which image the test image closely matches
    - Calculating Euclidean differences between the dataset weights and the test image weights
    - Finding the minimum difference
- For extra information…
    - Plotting the CMC curve

Procedure

Since this is the first project of the semester, I'm going to take a much different approach to explaining my thought process because my approach to the problem was much larger and a little more complicated than the basic homework assignments. For this report, I will explain each step in a little more detail to demonstrate what I've learned throughout this assignment.

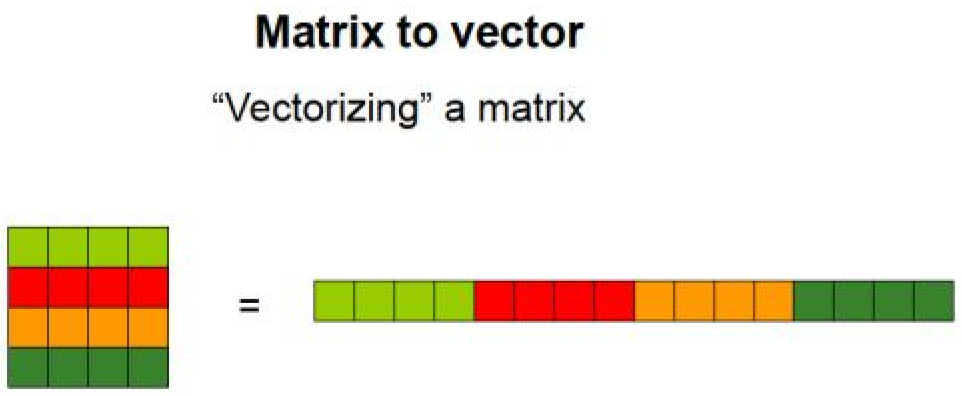
First of all, waiting until after spring break to start this project was a mess! I focused on another big paper to do instead of working on this project. So that was unfortunate, but I still managed to complete this project in a semi-timely manner.

PCA Training

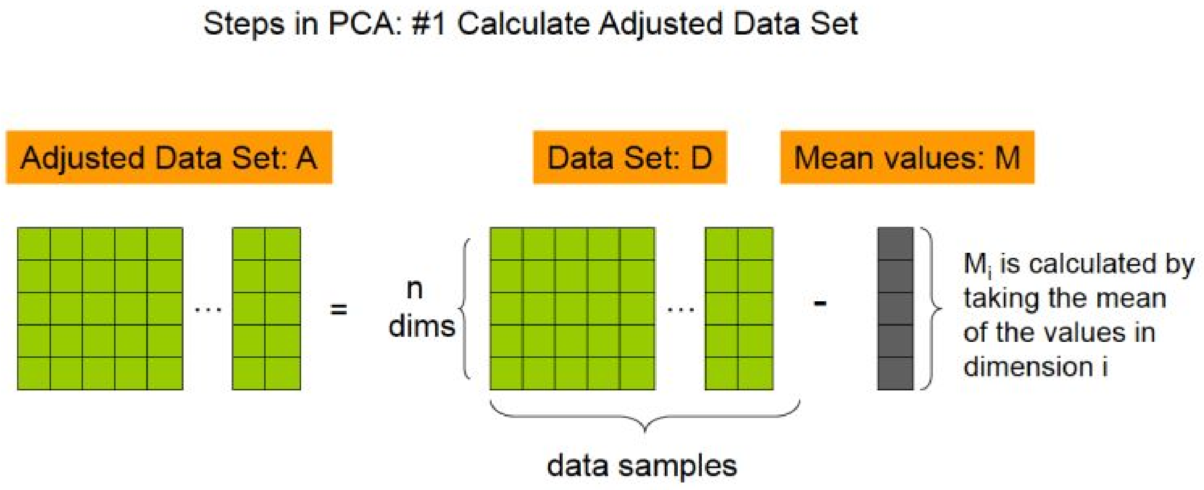
When I was watching several YouTube tutorials, reading several scholarly articles and papers, and reviewing Dr. Jiang's the following words repeatedly occurred: Vectorize, mean image, covariance, eigenvector, eigenvalue, eigenface, and weight. After much studying, I figured out that Principal Component Analysis (PCA) is not exclusive to image processing. PCA can be used in voice recognition as well because the application is nearly identical.

For our project, we had to train our program to recognize faces, starting with a basic implementation of PCA. For the record, I attempted my own basic approach, just to have a much better understanding of how PCA works. After several hours of study and Dr. Jiang's help, I finally developed a decent, standard PCA face recognition program (that works, complete with CMC curve).

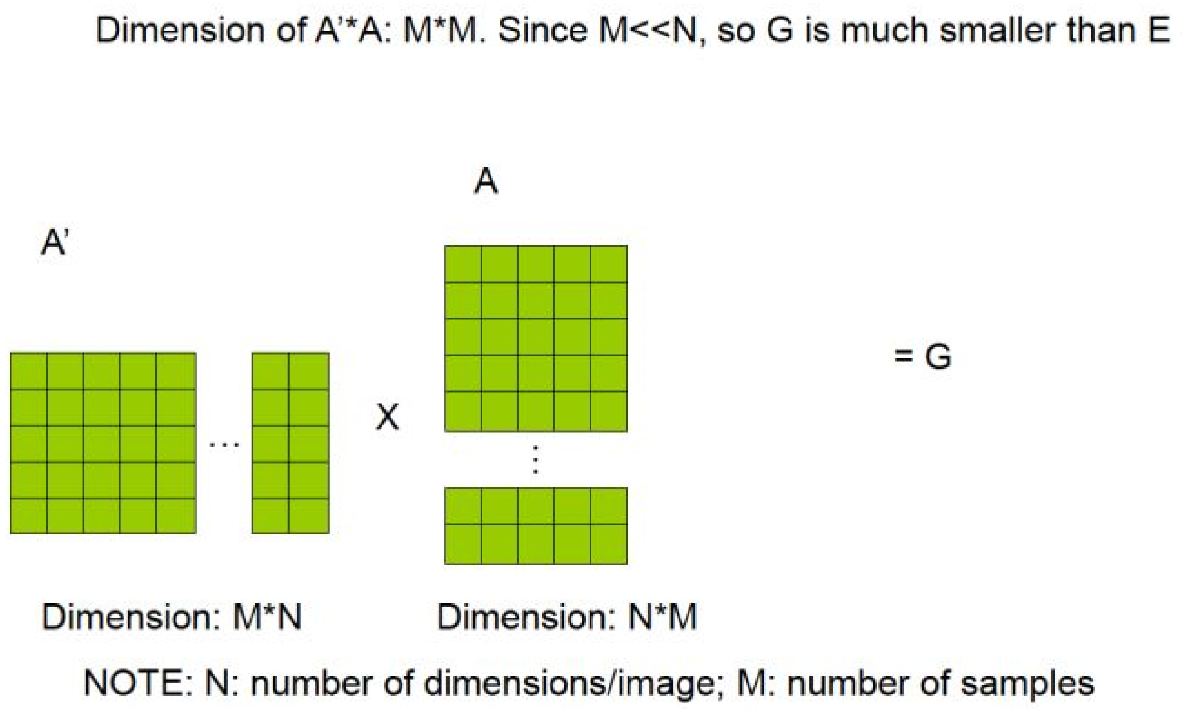
The first step was to fetch every image in the training dataset and vectorize the image like so.



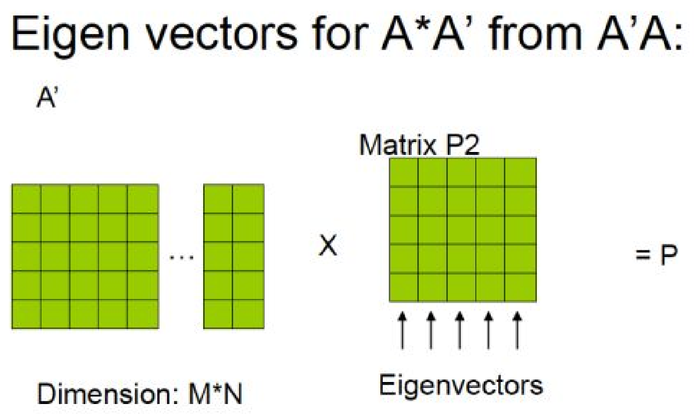
Simple enough. Next, I needed to calculate the mean image from each pixel value in each image and subtract the mean image from each image vector in the raw dataset.


Steps in PCA: #1 Calculate Adjusted Data Set
Adjusted Data Set: A
Data Set: D
Mean values: M
=
n dims
-
M_i is calculated by taking the mean of the values in dimension i
data samples

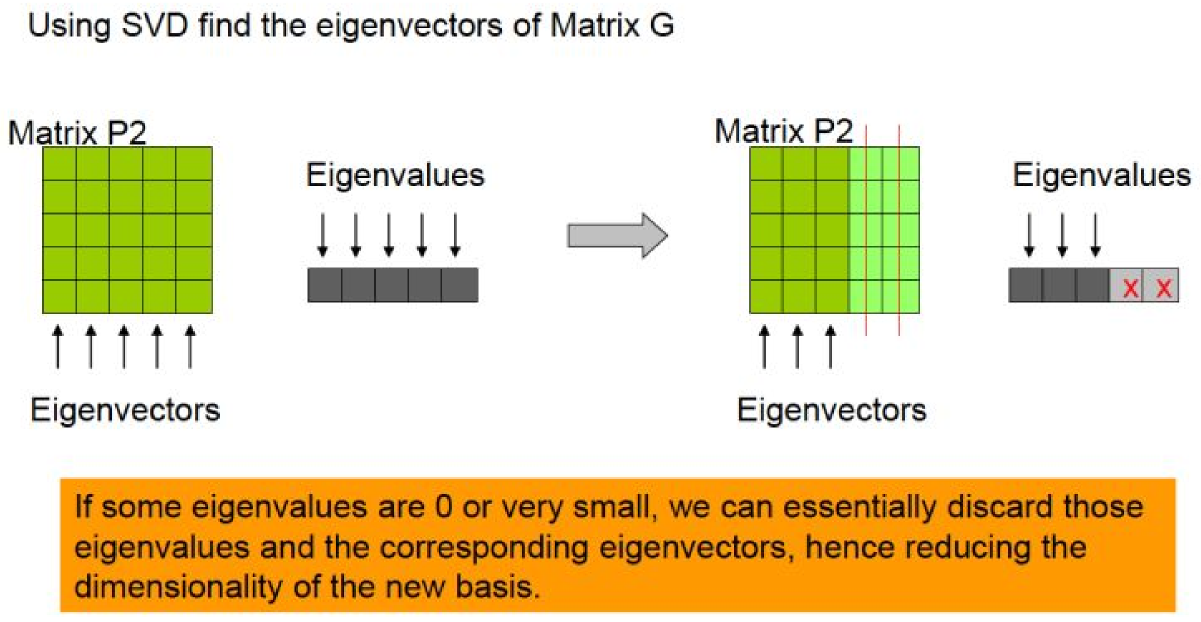
After getting the mean-subtracted dataset, the next step was to calculate the covariance matrix. This matrix was to be used for eigenvector, eigenvalue, and eigenface calculation later.


Dimension of A'*A: M*M. Since M<<N, so G is much smaller than E
A'
A
X
= G
Dimension: M*N
Dimension: N*M
NOTE: N: number of dimensions/image; M: number of samples

A very important concept I learned in this project is the mathematical concept behind what's known as "dimension reduction". For large datasets, this allows for *significantly* faster computational time. Instead of brute forcing each and every eigenvector (which would take a very long time), it was possible to extract the large eigenvector by computing the small covariance matrix, and multiplying the smaller covariance matrix by the smaller eigenvector matrix. This was the "version 2" PCA approach denoted in the pdf.



So the next step in the process was to calculate the eigenvectors and eigenvalues of the covariance matrix. The built-in function eig() comes in handy here.


Using SVD find the eigenvectors of Matrix G
Matrix P2
Eigenvalues
Eigenvectors
Matrix P2
Eigenvalues
Eigenvectors
If some eigenvalues are 0 or very small, we can essentially discard those eigenvalues and the corresponding eigenvectors, hence reducing the dimensionality of the new basis.

Regarding the large note in orange, I did not take that approach. I accounted for every eigenvalue even if it was minuscule. I did this simply for consistency. I prefer getting results that are as accurate as reasonably possible.

The last step was to calculate the weights of the training dataset by multiplying the transpose of the eigenvectors by the covariance matrix (both in reduced dimensionality). At this point, the training is done, and the testing may commence.

Testing

Testing was arguably the easiest part of the project. Testing a single image would have been satisfactory for the project, but if somebody wants to test multiple images, then it’s possible with my approach. My program will test each image under the “testing” folder against the dataset.

The first step in testing the images was to vectorize the image, as the system did in the training.

The next step was to create a mean-subtracted image vector from the already existing mean image from the training dataset. This allowed me to determine what unique features were in the test image based on what the system already knew from the training.

Next, I calculated the weights of the test image against the training dataset.

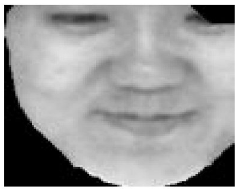
Since PCA typically uses euclidean differences to determine face matches against the dataset, the last step was to mathematically measure the euclidean distances between the testing image weights and the dataset weights. The minimum values indicate a matched image. Many PCA implementations have a “threshold” which can determine if the “match” is actually in the original dataset, but I did not implement this threshold feature in my code, mainly due to time constraints. So essentially, a picture of a car would match to some face in my dataset, but that’s a topic for another time.

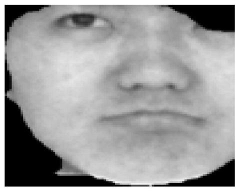
Analysis

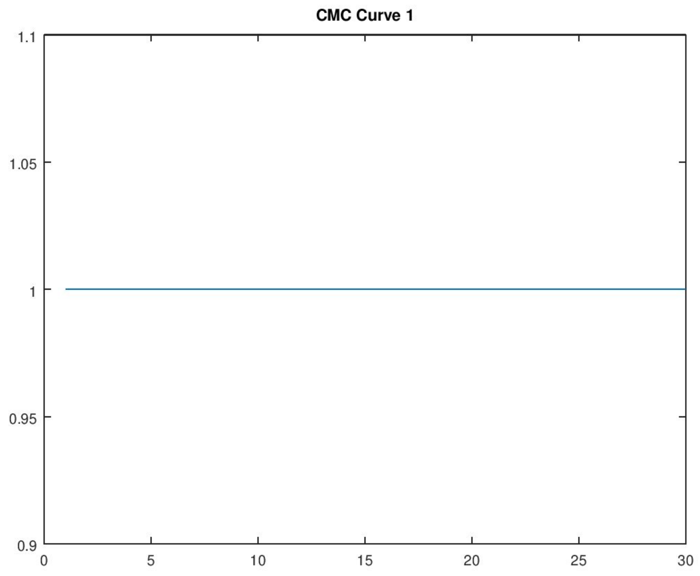
I tested 4 different images against the dataset. 3 of them were successful. I read somewhere that PCA typically has an accuracy of roughly 70%, which is “good enough”, but not for highly sensitive scenarios. For this project, though, it was good enough.

NOTE: The left image is the test image, and the right image is the matched image.

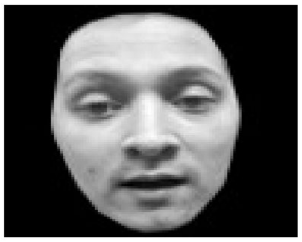
Test 1: Dr. Jiang (Is this you?)

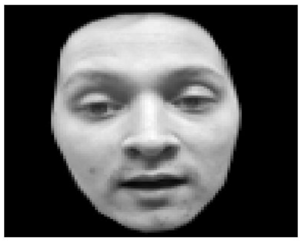


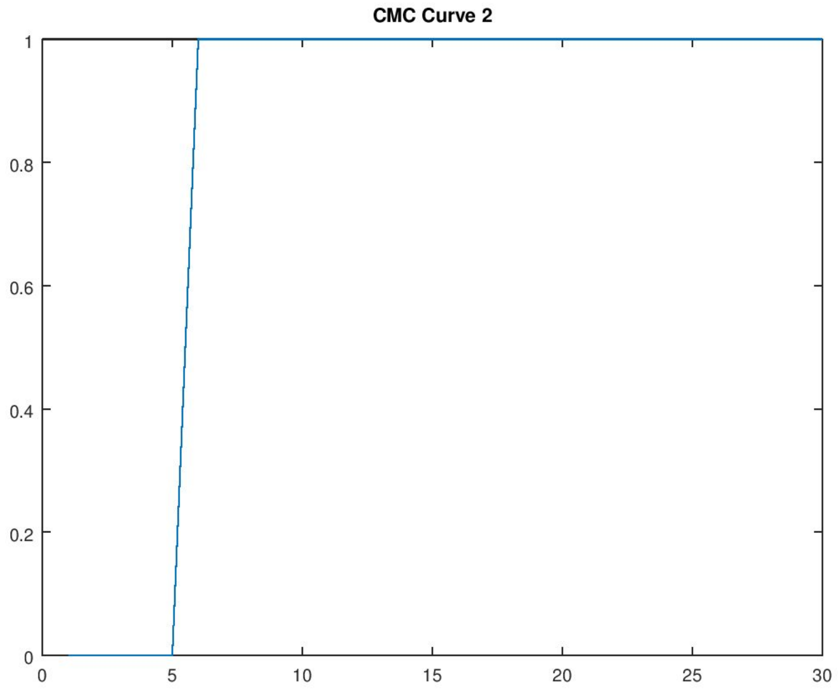




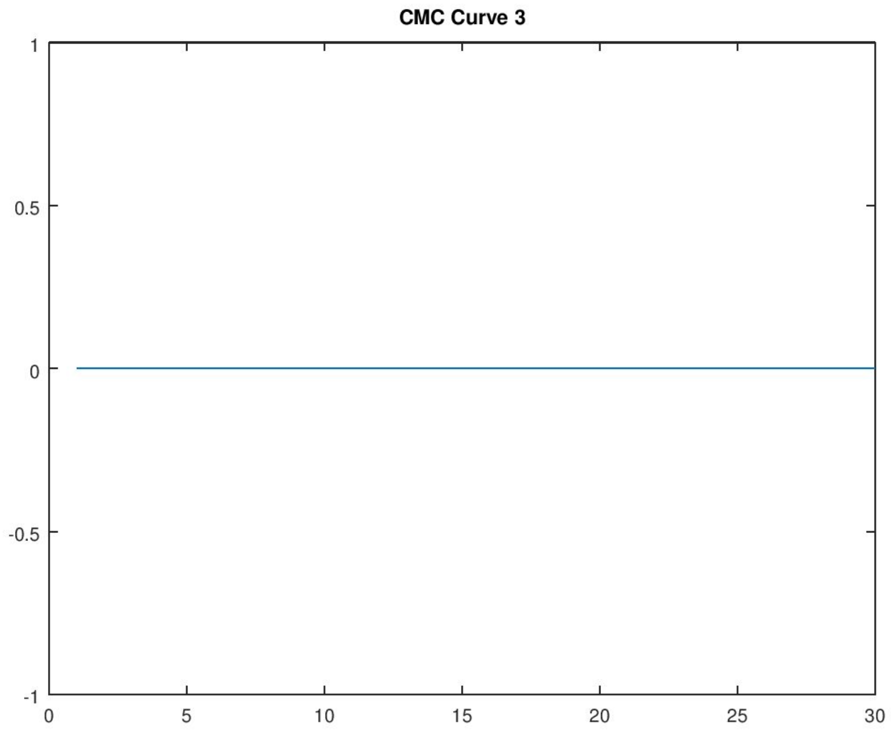
<u>Test 2: Myself</u>



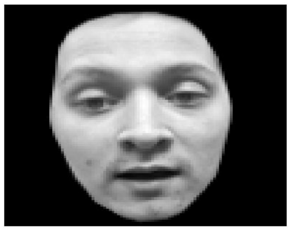


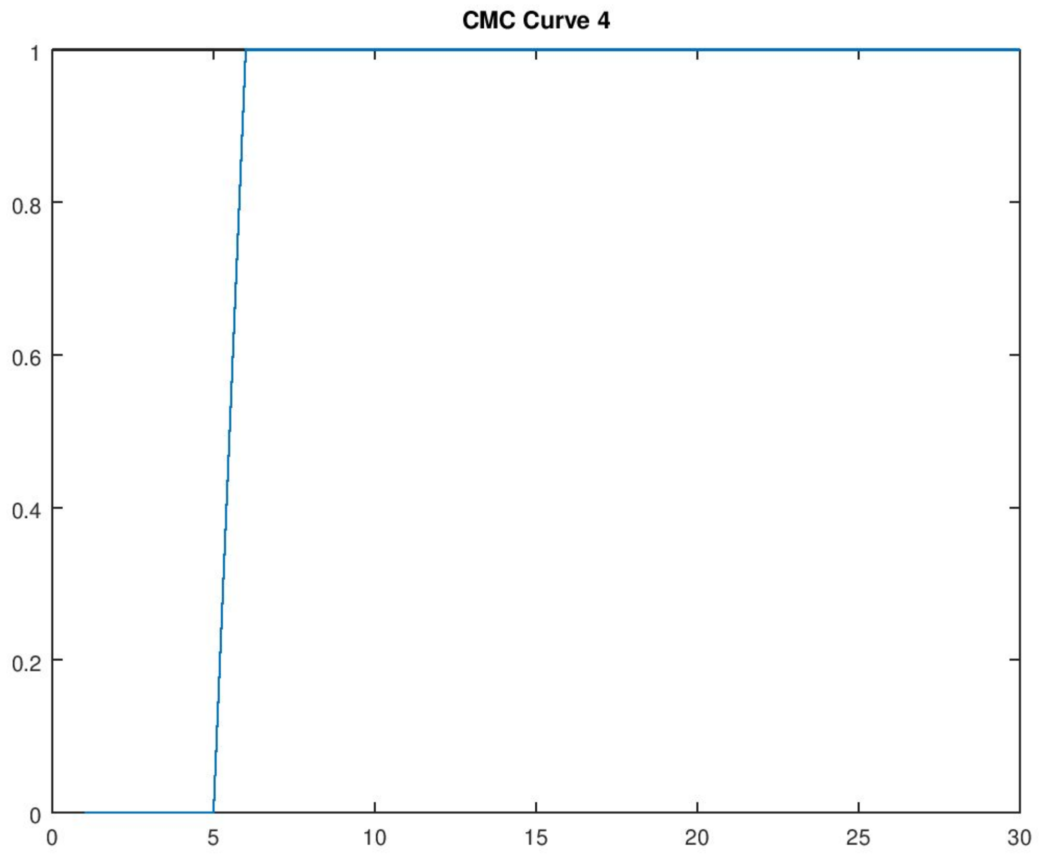


Test 3: Some dude from the dataset (Failure)



Test 4: Myself again





Conclusion

This project was a very good challenge. I almost gave up, but decided to get help from the master, Dr. Jiang. So with a little bit of YouTube, Internet magic, and the master's help, I was able to successfully assemble a functional PCA face recognition project.

Could it use some refining? Absolutely. For having no prior experience with PCA face recognition, I'd say I came out alright.

It's also interesting to discover that PCA is a technique used in other applications as well, like voice recognition. In my research, that was a very helpful tip for this project as well, even though voice signals and images are not related much.

Overall, the project was a good experience even though I should have at least started it over spring break. I'm very thankful for the help I received from the professor, because that allowed me to finish without giving up. That goes for any class though; everybody should know that nearly all professors are more than willing to help students directly. So thank you, professor!