Yale Faces and Music Classification

Ron Wilson

March 6, 2020

Abstract

Methods of SVD analysis are explored in two separate parts of this write-up. The first part is methods of image recognition, and the second part is an attempt to get an accurate music classification method.

1 Introduction and Overview

In the first part, I will use data from Yale Faces B to perform an SVD analysis on. There are two different data sets; the first one contains cropped faces, while the second one has the original, uncropped faces. Using the results of the SVD analysis, I will find the number of modes necessary to produce a good image reconstruction.

In the second part, there are three different tests to try to classify music. All three tests use 5 second audio clips of different songs. Test 1 attempts to classify three different artists (using three songs, one for each artist) of different genres. In test 2, I will take three artists of the same genre to try to classify the artist. Test 3 will take clips from different artists across three different genres. So, it will have three songs (from three different artists) for each of the three genres.

2 Theoretical Background

The Singular Value Decomposition (SVD) of an mxn matrix A is defined as:

$$A = U\Sigma V^*. \tag{1}$$

In the context of image recognition, the matrix A contains n columns where each column represents the data from one image. By computing the SVD, we get the mxn matrix U, which is the face space where the columns of U are the new bases of the data from SVD. The matrix Σ is an nxn diagonal matrix, containing the n singular values in descending order. The first r singular values that are the most significant is the rank of the face space. These r modes are found by diagonalizing the matrix Σ and plotting the results. Finally, the nxn matrix V represents the signature of the images. To reconstruct the images, we can use the rank r found from the SVD to perform the following operation:

$$B = U * \Sigma(:, 1:r) * V^{'}(:, 1:r). \tag{2}$$

Here, we are taking only the r necessary modes from Σ and the corresponding image signatures from the V matrix.

For music classification, the SVD is also very useful. However, here the matrix A will be the spectrogram data from the 5 second song clips. So the SVD will produce the matrix V which is the signature of the audio files. Using this signature, I can classify the music. I chose to use two different classification methods: k-nearest neighbors (KNN) and support vector machine (SVM).

KNN selects the k-nearest neighbors of a training set to classify the data. SVM makes a line to separate the training set into clusters, and classifies the clusters. The algorithms to set a training set and test the remaining data using these two classification methods can be found in section 3 of this write-up.

3 Algorithm Implementation and Development

Here I will explain the algorithms used for the two parts of this write-up. Please refer to Appendices A & B for the MATLAB functions mentioned in this section & the code corresponding to these algorithms.

3.1 Part 1: Yale Faces B

The goal in part 1 is to perform an SVD analysis to find the number of modes necessary to generate a good image reconstruction of the Yale faces. Here are the steps that I used to accomplish this (note that the algorithm is the same for both the cropped faces and the original faces):

1. To load the data, I used a for loop to change directories to access each of the sub-folders of images. Using the MATLAB function `imread`, I get the data from each image within the sub-folders. I then reshaped the data into a column vector and stored the data from each image as columns in a matrix (so each column in the matrix represents a different image).

2. Next, I computed the SVD of the data matrix. However, I converted the data matrix using the MATLAB function `double` to get into the correct input type for SVD. Also, I computed the reduced SVD to make the code run more efficiently. The reduced SVD removes the extra rows or columns of zeros from the matrix Σ and the corresponding columns in U and V that they would be multiplied by. This improves the execution time of the code without losing the accuracy of the SVD.

3. Now, I diagonalized the matrix Σ to produce the singular values. Dividing this diagonalization by its sum and multiplying by 100 gives the percentage of the singular values. Plotting these values yields the singular value spectrum (SVS), which shows us which modes are the most important in representing the data of the images. See figure 1 for the SVS of the cropped images, and see figure 2 for the SVS of the original images.

4. Finally, I reconstructed the images by computing the matrix B from equation 2 using the necessary number of modes r found from the singular value spectrum. I tested the reconstructions by plotting random images versus their reconstructed image. I also tested using different numbers of modes necessary by trying different values of r and plotting the reconstruction of a random image. See section 4 for the results of this SVD analysis on the two data sets.

3.2 Part 2: Music Classification

For part 2, we need to classify music genres and find the accuracy of the classifications. Here is the algorithm used to do this (note that this algorithm is the same for all three tests):

1. I loaded the data in the same way that I did for part 1 by changing directories within a for loop. However, this time the audio files for each clip contains two columns of data. So I took the mean of

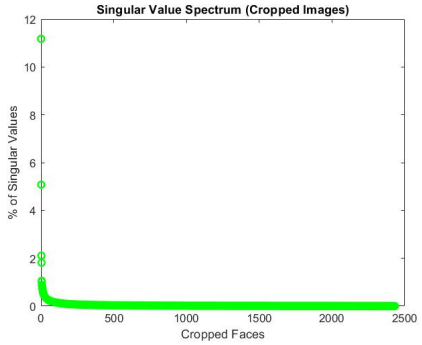

Figure 1: SVS for the Cropped Faces

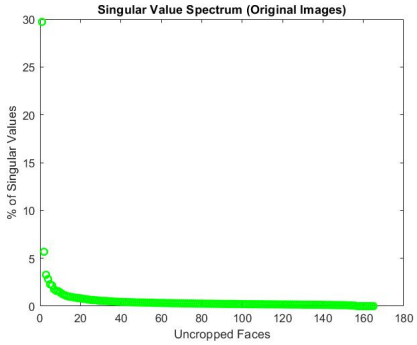

Figure 2: SVS for the Original Faces

the two columns to get a single column vector representing the data from each clip. Then I stored the data as columns in a data matrix for all of the clips.

2. Next, using the MATLAB function `randi`, I randomly generated a test set of 10 clips and a training set with the rest of the clips.

3. Once again, I compute the SVD. However, the SVD needs to be computed on the spectrogram matrix rather than the data matrix since the spectrogram represents the frequencies of the audio from the clips. So I used a for loop to take the FFT of each clip and store the frequency data in a spectrogram matrix. Note that I also need to subtract the mean from the specrogram matrix before computing the SVD to make sure that the data is centered for all of the clips.

4. Now, I can classify the music using KNN and SVM. The matrix V from SVD is used because it contains the signature of the audio clips. The MATLAB functions `fitcknn` (KNN) & `fitcecoc` (SVM) produce a model for the classification of the signatures from V and the training set. Plugging this model into the function `predict` yields the classification labels to apply to the testing set.

5. Finally, I used a for loop to count the number of accurate classifications in the testing set. By dividing this number by the length of the testing set and multiplying by 100, I get the percentage of the accuracy of the classification. I then plotted a bar graph for each of the three tests to compare the accuracy of KNN versus SVM. See section 4 for the results of each test.

4 Computational Results

4.1 Part 1: Yale Faces B

From the singular value spectrum of the cropped faces (see figure 1) we can see that the first 4 modes are the most important. Thus, the rank of the face space is $r = 4$, which is approximately 20% of the singular values. Similarly, the rank of the face space for the original faces is $r = 6$ (see figure 2), which makes up approximately 46% of the singular values. These values for r are the number of necessary modes needed for good image reconstructions.

Using these values, I reconstructed four random images for each of the two data sets. See figure 3 for the cropped images and see figure 4 for the original images. For the cropped images, the reconstruction was able to capture the basic facial structure as well as the lighting of the image, but the reconstructed face looks like the same person across all four images. Meanwhile, the reconstructions of the original images are clearly different people and it captures the structure and lighting as well. However, the original image reco-

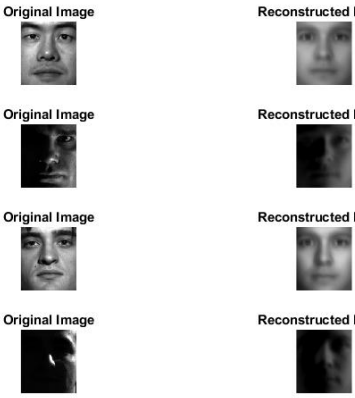

Figure 3: Cropped Images (Rank r=4)

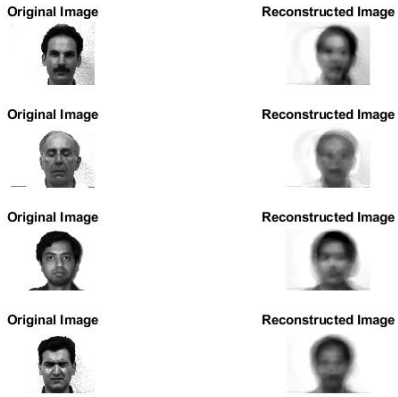

Figure 4: Original Images (Rank r=6)

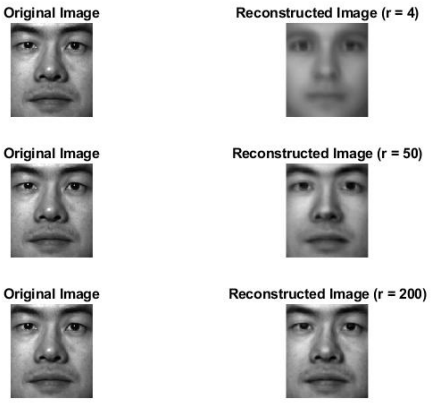

Figure 5: Cropped Face Using Different Ranks

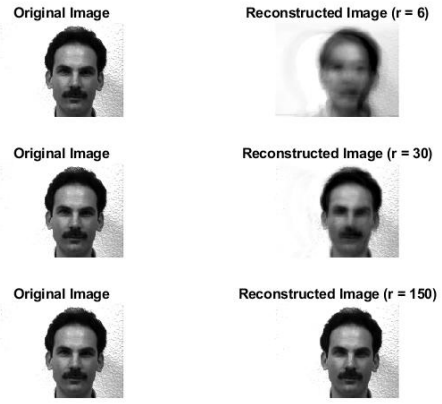

Figure 6: Original Face Using Different Ranks

nstructions are very blurry images. This is likely because the images were not cropped, so the faces are in different positions across all of the data, which makes it very difficult to get a good reconstruction.

The values for the rank of the face space used were chosen to be the smallest values that still yield a good reconstruction. However, it is also useful to look at using larger values. I took the first image from the four random images and tried three different values for the rank. For the cropped faces (see figure 5) I used $r = 4$, 50, & 200. For the original faces (see figure 6) I used $r = 6$, 30, & 150. Notice that the reconstruction becomes much more accurate using higher rank approximations for both data sets. However, it is not always feasible to use high-rank approximations as it takes more memory and time to execute the code.

4.2 Part 2: Music Classification

In test 1, I used three songs from three different artists of three different genres (so a total of three songs). See figure 7 for the accuracy's of the KNN and SVM classifications. KNN had an accuracy of 40%, while SVM was 70% accurate. It is expected that SVM was more accurate than KNN, however I did not expect KNN to have such a low accuracy. This may be because the songs are all different lengths and thus there is a different number of 5 second clips per song. So if the randomly generated training set doesn't include as many clips of a particular song, it could make it difficult to classify the testing set. Meanwhile, the SVM is

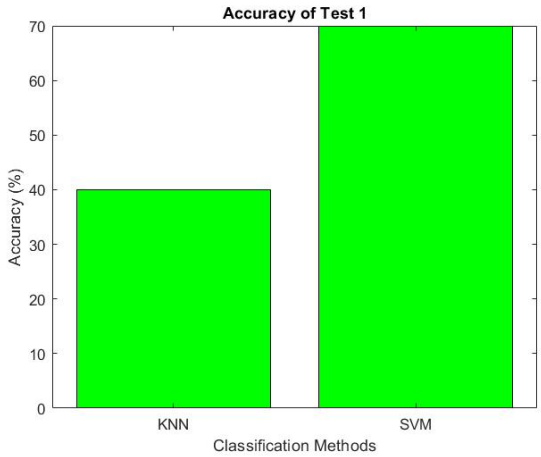

Figure 7: Test 1 Accuracy using KNN and SVM Classifications

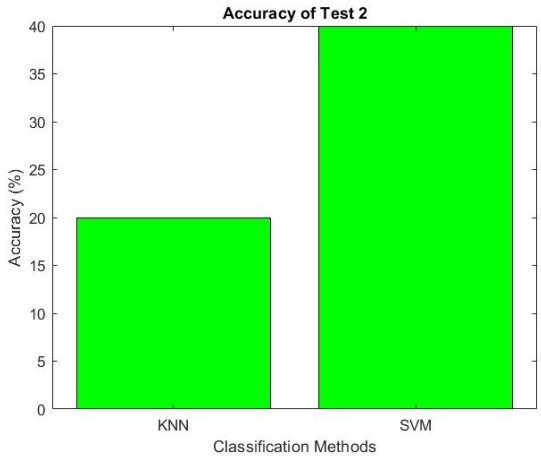

Figure 8: Test 2 Accuracy using KNN and SVM Classifications

a lot better at clustering, so the different amounts of clips isn't as much of an issue.

For test 2, I took three songs from three different artists of the same genre. See figure 8 for the accuracy's of this test. Here, KNN was 20% accurate and SVM was 40% accurate, which is nearly half of the accuracy for both methods from test 1. This is expected as the song clips are all within the same genre, which should make it very difficult to classify. Still, the SVM method was slightly more accurate, but 40% is definitely not enough for a reliable classification.

Finally, in test 3, I used three songs per genre from three different artists (so a total of nine songs). See figure 9 for the accuracy's of test 3. This test had a very long processing time because of the much larger amount of song clips to process. I would expect the results to be more accurate for both classification methods than what test 1 yielded because of the larger training set. However, one issue is that the three different artists from the same genre might not sound exactly the same, which would lead to a less accurate classification. This may be an issue with my song selection because the results were not more accurate than in test 1. KNN and SVM both had accuracy's of 40%. This is the same as the test 1 accuracy for KNN, but SVM was less accurate. It is also possible that the data set here is too large for KNN and SVM to produce accurate classifications and maybe there is a better method for such large data.

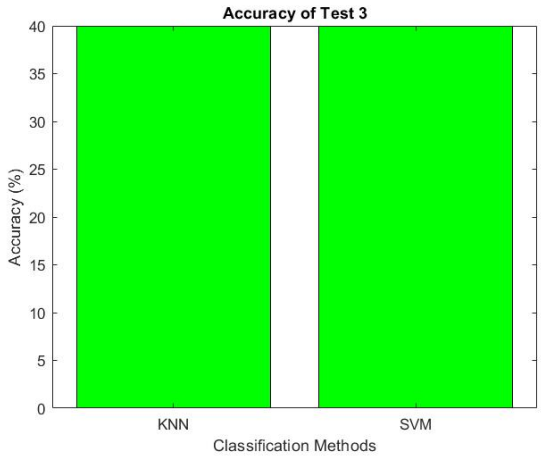

Figure 9: Test 3 Accuracy using KNN and SVM Classifications

5 Summary and Conclusions

The SVD is a very powerful tool to use to analyze large amounts of data. In part 1, I was able to get a really good image reconstruction for both cropped and uncropped images. In part 2, I was able to use the signature matrix from SVD to classify music genres. However, it is clear that the classification methods KNN and SVM might not be the best for large data.

Appendix A MATLAB Functions

The following is a list of the important MATLAB functions that were used:

- `listing = dir(name)` returns attributes about `name`.
- `cd(newFolder)` changes the current folder to `newFolder`.
- `A = imread(filename)` reads the image from the file specified by `filename`.
- `B = reshape(A,sz1,...,szn)` reshapes `A` into a `sz1-by-...-by-szN` array where `sz1,...,szN` indicates the size of each dimension.
- `Y = double(X)` converts the values in `X` to double precision.
- `[U,S,V] = svd(A,'econ')` produces an economy-size singular value decomposition of matrix `A`, such that `A = U*S*V'`.
- `imshow(I)` displays the image `I` in a figure.
- `[y,Fs] = audioread(filename)` reads data from the file named filename, and returns sampled data, `y`, and a sample rate for that data, `Fs`.
- `X = randi([imin,imax],sz1,...,szN)` returns an `sz1-by-...-by-szN` array containing integers drawn from the discrete uniform distribution on the interval `[imin,imax]`, where `sz1,...,szN` indicates the size of each dimension.
- `Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `B = repmat(A,r1,...,rN)` specifies a list of scalars, `r1,..,rN`, that describes how copies of `A` are arranged in each dimension. When `A` has `N` dimensions, the size of `B` is `size(A).*[r1...rN]`.
- `Md1 = fitcknn(X,Y,Name,Value)` returns a k-nearest neighbor classification model based on the predictor data `X` and response `Y`, and fits the model with additional options specified by one or more name-value pair arguments.
- `label = predict(Md1,X)` returns a vector of predicted class labels for the predictor data in the table or matrix `X`, based on the trained discriminant analysis classification model `Mdl`.
- `Md1 = fitcecoc(X,Y)` returns a trained ECOC model using the predictors `X` and the class labels `Y`.
- `bar(x,y,color)` creates a bar graph with one bar for each element in `y`, draws the bars at the locations specified by `x`, and sets the color for all the bars.

Appendix B MATLAB Code

The MATLAB code used to generate the results of this write-up (`Yale_Faces_and_Music_Classification.m`) can be found at the following GitHub repository: `https://github.com/ronwilson016/Data-Science-Projects/tree/master/Machine%20Learning/Yale%20Faces%20and%20Music%20Classification/MATLAB%20Code`