AMATH 482 HW 4

Use of the Singular Value Decomposition in Classification

Zachary McNulty
zmcnulty, ID: 1636402

Abstract: In this paper, we will explore the power and limitations of the SVD in extracting fundamental structure from data. To do so, we will attempt to form a low-rank approximation of the human face and explore the consequences of using data that has not been preprocessed for SVD. Furthermore, we will use these ideas in order to develop a coordinate system ideal for classification via Support Vector Machines (SVM), k-Nearest Neighbors (KNN), and Decision Tree Learning (DTL). As an example, we will run this algorithm on a series of music samples from different bands and genres and analyze their performace.

University of Washington
March 2019

1 Introduction and Overview

The SVD is a powerful tool for understanding the fundamental structure of data, but it has certain limitations. One of these limitations is that it does not handle translated data well. Because the SVD relies on simple linear combinations of its principal modes to capture features in the data, shifted data measurements cannot be well-captured by their unshifted counterparts. This can lead to an artificially high approximation of the dimension of the system, reducing the low-rank benefits SVD provides. To highlight this shortcoming, we will perform SVD analysis on two groups of images from the Yale Faces Database: preprocessed, cropped headshots and uncropped, raw headshots. Later on in this report, we will use what we learned about the SVD to develop powerful classification techniques. Specifically, we will use SVD to find a low-dimension representation of our data and choose some of these dimensions, those which show the most variability between the groups we are trying to classify, to form a coordinate system for classification. Once we have developed this coordinate system, we will use Support Vector Machines (SVM), k-Nearest Neighbors (KNN), and Decision Tree Learning (DCL) to perform the classification, analyzing the results of each of these three models.

2 Theoretical Background

The Singular Value Decomposition is a diagonalization of a given matrix A that focuses on the rotations/reflections and stretching a vector undergoes when transformed by A. Formally, for any matrix $A \in \mathbb{C}^{m\times n}$ there exists a diagonalization of the form:

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

where $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary matrices, $\Sigma \in \mathbb{R}^{m\times n}_{\geq 0}$ is a diagonal matrix, and V^* represents the complex conjugate transpose of V. As U and V are unitary matrices their columns each form an orthonormal basis for their respective space and the matrices have the convenient property that $UU^* = U^*U = I_m$ and $VV^* = V^*V = I_n$. The diagonal entries of Σ are called the **singular values** of A. Intuitively, this breaks down the transformation A into three fundamental transformations: a rotation/reflection V^* within the the domain of A, a scaling of the components of the space X by Σ, and another rotation/reflection of the space within the codomain of A. To understand the importance of the SVD, first consider the covariance matrix for a data matrix X:

$$C_X = \frac{1}{n-1}XX^T \tag{2}$$

In this matrix, entry (i, j) corresponds to the covariance between row i and row j of X (data measurement i and j). If $i \neq j$, then a high value in entry (i, j) suggests the two data measurements vary in similar ways and are thus likely redundant measurements of the same feature in the data. Consequentially, a low value at (i, j) suggests the two data measurements are fairly independent. If $i = j$, the diagonal of C_X, then entry (i, j) describes the variance in data measurement i. Generally, data measurements with a high variance are assumed to capture important features of the data while those with low variance do not. To see the importance of the SVD, consider the covariance matrix for $Y = U^*X = \Sigma V^*$ where U, Σ, V are from the SVD of X:

$$C_Y = \frac{1}{n-1}YY^T = \frac{1}{n-1}\Sigma^2 \tag{3}$$

As C_Y is diagonal, all its off diagonal entries are zero: projecting the data X onto U^* produces a transformed data set ΣV^* with completely independent measurements, eliminating redundancy in the data. Similarly, if we chose $Y = XV = U\Sigma$ we find the same covariance matrix, showing us $U\Sigma$ also forms an independent set. As the columns of U and V form bases for the codomain and domain of X respectively, each give an ideal, independent coordinate system for each space. If the data samples in X are stored as rows, V gives the ideal coordinate system and U the coordinates of each sample in that system. Else, if the samples are stored as columns then U gives the ideal coordinate system and V the coordinates of each sample in that system. Furthermore, the corresponding singular value σ in the diagonal of Σ ranks the importance of each direction in this coordinate system based on the data's variance along that direction. Note for any given matrix X we can reconstruct it given these coordinates:

$$X = \sum_{i=1}^{min(m,n)} u_j\sigma_j v_j^* \tag{4}$$

where u_j, σ_j, v_j are the jth column of U, Σ, V respectively. Since the singular values σ_j are a non-increasing sequence, we see each successive term is increasingly less important to the structure of X. Thus, if the singular values decrease rapidly, we can get a good approximation of X with only the first few terms of this sum. This is a low-rank approximation of the matrix X, a simplified form of the dynamics of the system that well-captures its behavior.

As we will show later, the SVD is very sensitive to translations. In this regard, we can use tools such as the Fourier Transform to move the data into a space where this translation is not as relevant. For example, if we were to have two identical sinusoidal signals shifted completely out of phase, SVD on the time series data would give two prominent modes. However, performing the SVD in the frequency domain on the Fourier Transforms of these signals will yield a single prominent mode: the frequency of the system. The Fourier Transform is the primary tool for converting between the spatial and frequency domains of a signal. Below is the transform and its inverse:

$$F(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \tag{5}$$

$$f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{ikx} F(k) dk \tag{6}$$

Where k is the wavenumber. It may not be clear from **Equations 5** or **6** why this encodes frequency information, but expanding e^{-ikx} provides some insight:

$$e^{-ikx} = \cos(kx) - i\sin(kx) \tag{7}$$

Equation 7 makes it clear why then k represents the wavenumber. Thus the transform is essentially taking the inner product of our function with a bunch of periodic sines and cosines where, roughly speaking, frequencies/wavenumbers more similar to our signal get assigned higher values. In this way, spatial information is converted to the frequency domain.

Simplifying the system allows us to extract the important characteristics of a system. Not only does this reduce the amount of computational work required to work with the system, but it lowers the number of parameters involved and makes overfitting a training algorithm less likely. The three supervised learning algorithms used in this paper are KNN, SVM, and DTL. Supervised learning follows a simple idea: given a set of labeled data measurements from a set of classes, learn to classify future data measurements into one of the pre-defined classes. KNN simply classifies new points by finding which of the labeled points the new point is closest to (or more generally, of the K nearest labeled points, which class is most common). SVM is essentially an optimization problem that tries to find the an optimal hyperplane $\vec{w}x + b = 0$ where all points x satisfying $\vec{w}x + b < 0$ are labeled as class one and all points satisfying $\vec{w}x + b > 0$ are labeled as class two. In this case, the optimal hyperplane is defined as the one in which there are as few errors as possible in classification on the training data and that distance separating the classes around the boundary is as large as possible. The latter constraint tries to make it less likely the variability in the input will cause a change in classification. Lastly, DTL follows a simple binary approach. For each variable in our system, scan over all possible values of that variable to be use as a separator of that data. Find the variable and its associated value that best divides the current collection of data into their respective groups. Divide the data according to this variable/value and repeat the process for both halves of the data, continuing for a set number of iterations or until some tolerance is reached. In the end, the algorithm yields a series of conditions that can be used to classify the data by following the path down the decision tree. As the success of all three of these supervised learning algorithms depends on how well they can separate the classes within the coordinate system they are working in, SVD can be very important for helping develop the proper space to work in.

3 Algorithm Implementation and Development

3.1 Yale Faces

1. Loop over all the folders containing photos for each subject (Appendix B: code 0-31).
2. For each folder/subject, take each photo, reshape it into a column vector, and place it as a column in our data matrix F. Additionally, create an average of all the images for each subject and store it as a column in F_{ave} (Appendix B: code 35-50).

3. Mean subtract from data matrices F and F_{ave} and perform SVD (Appendix B: code 59-71).
4. Plot the singular values, analyze the principal components (columns of U), and choose a rank r that captures at least 90% of the energy of the system (Appendix B: code 97-139).
5. Perform the rank r approximation of the faces and analyze the effectiveness of the approximation (Appendix B: code 145-163).
6. Repeat for the uncropped images and compare results (Appendix B: code 164-307).

3.2 Music Classification

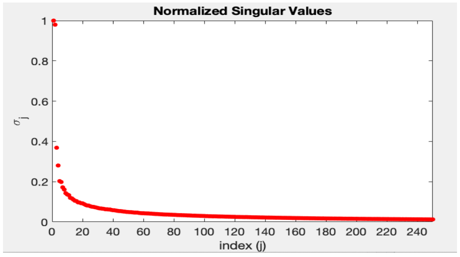
1. Loop through the folders for each group (Appendix B: code 0-42).
2. For each group, loop through all the songs and take 5-second samples with a random starting point, storing which group the samples are from. We ignore the first 30 seconds of a song to avoid sampling the period of time before the music begins (Appendix B: code 49-116).
3. Perform the FFT on each sample to convert to frequency domain. Place half these samples into the training data set and the other half into the validation dataset (Appendix B: code 81-92).
4. Mean subtract from each sample and perform the SVD on the entire training dataset (Appendix B: code 133-134).
5. Plot singular values and determine principal components that adequately separate the data to develop a better coordinate system for classification. Choose a rank that captures at least 90% of the energy of the system (Appendix B: code 137-158).
6. Train SVM, KNN, and DTL models using training dataset, training the model on the coordinates of each of these pieces in the new coordinate system determined by SVD in the previous steps (coordinates of each music piece with respect to the basis U) (Appendix B: code 182-203).
7. Check accuracy of the model generated by projecting each piece of data in the validation dataset onto the basis/coordinate system determined by SVD and check the model's prediction (Appendix B: code 213-226).
8. Plot classifications and analyze where misclassifications occurred (Appendix B: code 233-242).
9. Repeat this process for all three test cases.

4 Computational Results

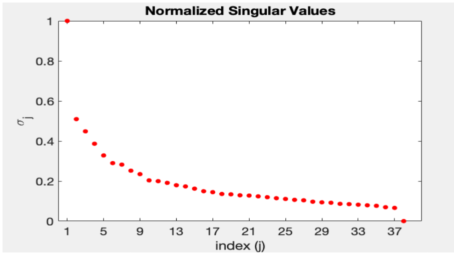
4.1 Yale Faces

Since we stored each image as a column in our data matrix F, the SVD has a standard interpretation. As U forms an orthonormal basis for the codomain of our data matrix F and the codomain is the space where each image exists, the U matrix is our coordinate system for "face-space". We can see this when we plot the columns of U which make of the basis of face-space. As we see in **Figure 1**, each principal component captures some of the general features of a face. Similarly, just by considering the matrix multiplication involved in $F = U\Sigma V^*$, we see that after scaling these principal components by Σ, each column of V^* gives the coordinates of each image, a column in F, in this face-space defined by U: Each face in F is a linear combination of the columns of U weighted by the columns of ΣV^*.

When we are working with the cropped images, we see that the averaging works quite well. As all the faces are aligned in the frame, averaging them serves to balance out the lighting in the scene. Furthermore, as we see in **Figure 2**, averaging allows us to drastically reduce the size of our system. While it appears that both systems have similar trends in the singular values, it takes 1181 modes to capture 90% of energy in the system versus only 29 modes in the averaged-face system. By reducing the dimension of our system, we can greatly improve the amount of computational work we can accomplish with it and reduce any overfitting that might come from



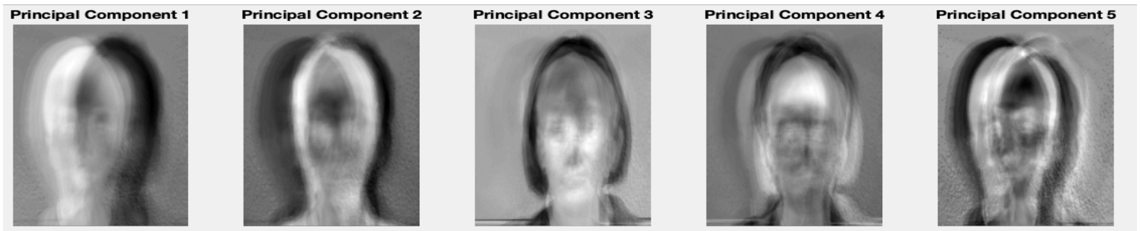
(a) Unaveraged Faces

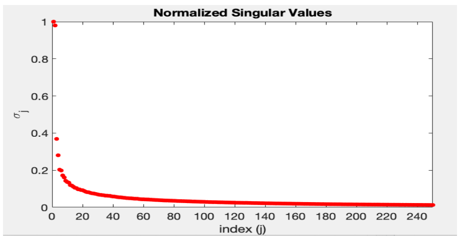


(b) Averaged Faces

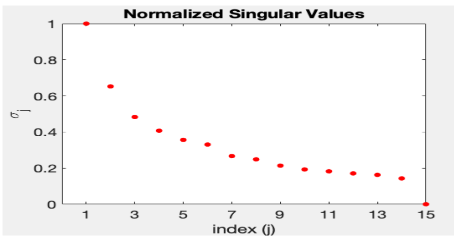
Figure 1: Face-space for the cropped, averaged images (top). The columns of U reshaped into the dimensions of the original images and plotted as an image. We can see each principal component captures some of the general features of a face. We can see there are many large singular values, suggesting face-space is fairly high-dimensional.

trying to extend this system, two key features whose importance will become clear in the Music Classification section. Conversely, with the uncropped images the averaged image does not make much sense. Since the subjects are in different locations, averaging the image introduces a large amount of blur and makes it difficult to distinguish key features of the face. Even though this reduces the dimensionality of our system, it is not clear the reduced system generated will be on any benefit. As we see in **Figure 3**, the face-space for the averaged, uncropped images is poorly recognizable. Even without averaging, the varying location of the subject makes it difficult for the SVD to pick out important features and we see a similar face-space. While the averaging in the uncropped face-space still allows us to reduce dimensionality, its cost is clearly seen in the singular values of **Figure 3**: there are many more relevant (high variance) modes than in the uncropped case. Thus, our low-rank approximations will not be as good and we lose some of the aforementioned benefits of our reduced system.





(a) Unaveraged Faces



(b) Averaged Faces

Figure 2: Face-space for the uncropped, averaged images (top). The columns of U reshaped into the dimensions of the original images and plotted as an image. We can see with the uncropped photos, the averaged image poorly captures features of face-space. This is also reflected in an increase in the decay rate of our singular values (bottom), clearly apparent in the averaged case.

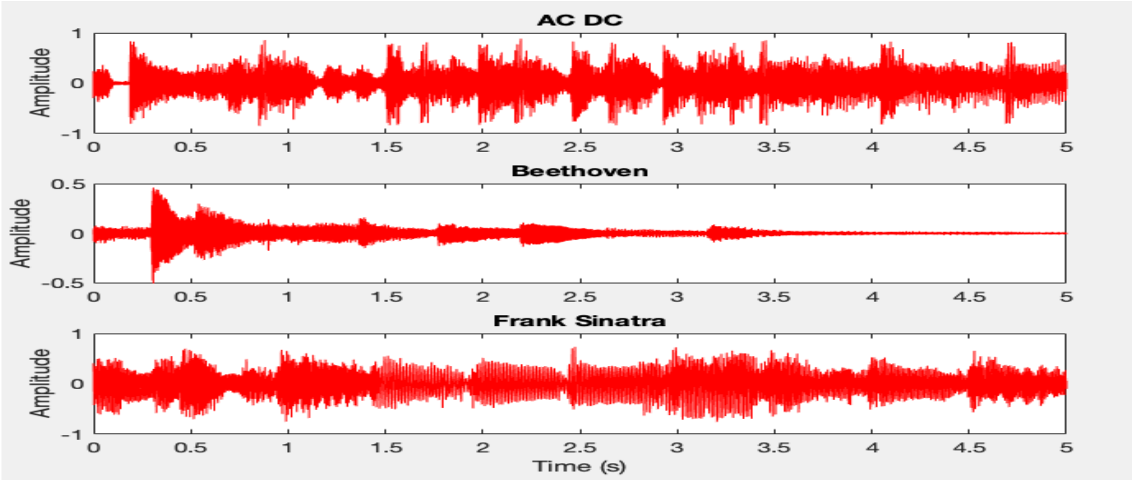


Figure 3: The music from different songs, bands, and genres each have a unique frequency pattern. As we see above, AC/DC has rapid frequency patterns which change suddenly, while Frank Sinatra and Beethoven have much more even patterns.

4.2 Music Classification

As we can see in **Figure 4**, each song, band, and genre have characteristic patterns that make them what they are. Using SVD, we can break down these patterns into a coordinate system that is more centered around the fundamental structures found in music. Rather than using the entire frequency spectrum for each piece of music, we can train our classification models using the coordinates given by this system. This will help us develop our classifiers by providing a low-dimensional space for them to train in.

4.2.1 Band Classification

In this test case, we tried to build a model to differentiate the music of three bands from different genres: AC/DC, Frank Sinatra, and Beehtoven. The goal was to determine which band each piece came from. Running our model, we saw that SVM was able to separate the training data fairly well. After generating the model, only around 10% of the training data was misclassified. However, we can see that this result does not generalize well as the model's accuracy on the testing data was only around 45%, slightly above random chance. DLT performed similar to SVM with a low training error rate, only 5%, but a low accuracy on the testing data, only 33%, suggesting this model also does not generalize well. On the other hand, KNN did a much better job at segregating the training data, developing a model which classifies every piece of data in the training set correctly. Futhermore, KNN had significantly bettter performace on the testing data set than either SVM or DLT, giving around 65 % accuracy. As we can see in **Figure 4**, SVM was fairly consistent, correctly classifying and misclassifying each class of music about the same percentage of the time, while KNN got a bulk of its correct classifications from class two (Beethoven). Interestingly, Decision Tree classifications almost never predicted case 2 (Beethoven) and had a large preference for predicting case 3 (Sinatra).

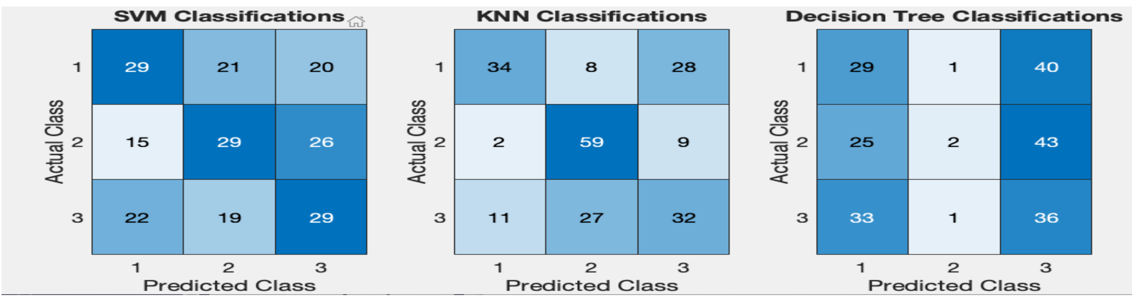


Figure 4: Case 1 classifications for each of the supervised learning algorithms. Entry (i, j) of each figure counts the number of test cases that were classified as band i and were actually band j. In this case, class one is AC/DC, class two is Beethoven, and class 3 is Sinatra.

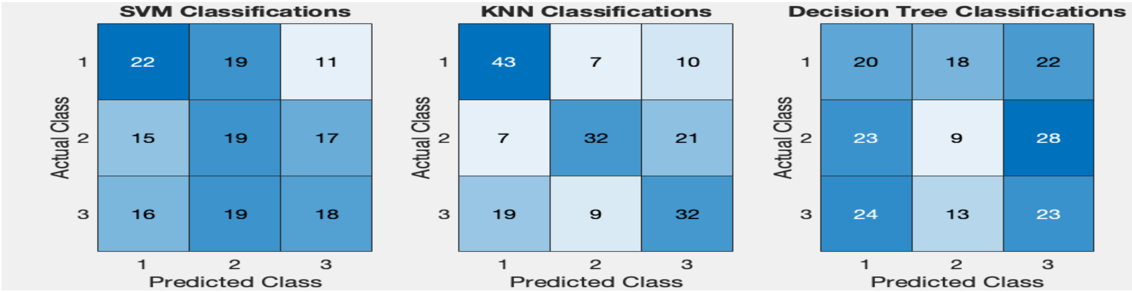


Figure 5: Case 2 classifications for each of the supervised learning algorithms. Entry (i, j) of each figure counts the number of test cases that were classified as band i and were actually band j. In this case, class one is Alice in Chains, class two is Pearl Jam, and class 3 is Sound Garden.

4.2.2 The Case For Seattle

In this test case, we tried to build a model to differentiate the music of three bands from the same genre, grunge/rock, and all from the Greater Seattle area: Alice in Chains, Pearl Jam, and Soundgarden. The goal was to determine which band each piece came from. Running our model, we saw that SVM was not able to separate the training data fairly well. After generating the model, over 35% of the training data was misclassified. However, we can see that unlike in the previous case, this result does generalize well: the model's accuracy is about 62 %, approximately the same as what it was on the training data. DLT performed better than SVM on the training data, only misclassifying 5% of the sample, but did roughly the same on the testing set with an accuracy of around 59%.Once again, KNN did a much better job at segregating the training data, developing a model which classifies every piece of data in the training set correctly. Futhermore, KNN had significantly bettter performace on the testing data set than either SVM or DLT, giving around 77 % accuracy. From **Figure 5**, we can see that once again SVM is much more consistent than the other two methods, correctly predicting and misclassifying each of the possible classes at the same rate. Furthermore, once again KNN is better at predicting some classes than the others, almost never misclassifying a piece in class one (Alice in Chains). In this test case, Decision Tree classification was much more consistent, although it seems to have particular difficultly classifying pieces from class two (Pearl Jam). Interestingly, the models performed better in this case than any of the others.

4.2.3 Genre Classification

In this test case, we tried to build a model to differentiate the music of a variety of bands from the same genre. The goal was to determine which genre each piece of music fell in to: classical (Bach, Mozart, Beethoven, Vivaldi), heavy metal (Metallica, Iron Maiden, Black Sabbath), or country (Johnny Cash, Zac Brown Band, and Merle Haggard). Running our model, we saw that SVM was able to separate the training data fairly well. After generating the model, only around 13% of the training data was misclassified. However, we can see that this result does not generalize well as the model's accuracy on the testing data was only around 43%, slightly above random chance. DLT performed similar to SVM with a low training error rate, only 5%, but a low accuracy on the testing data, only 34%, suggesting this model also does not generalize well. On the other hand, KNN did

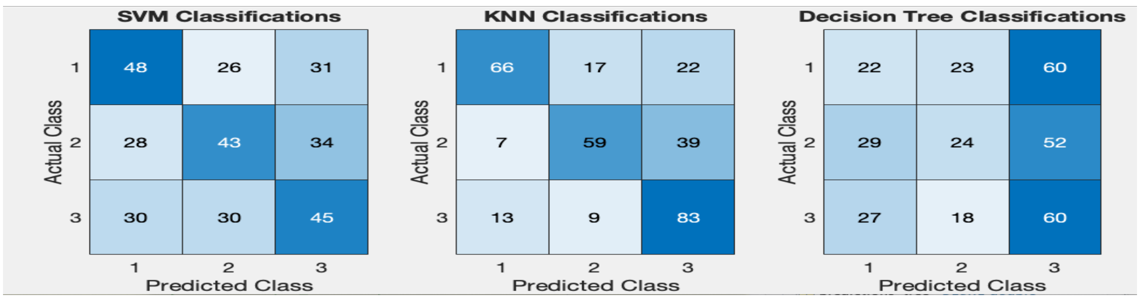


Figure 6: Case 3 classifications for each of the supervised learning algorithms. Entry (i, j) of each figure counts the number of test cases that were classified as band i and were actually band j. In this case, class one is classical music, class two is country music, and class 3 is heavy metal music.

a much better job at segregating the training data, developing a model which classifies every piece of data in the training set correctly. Futhermore, KNN had significantly bettter performace on the testing data set than either SVM or DLT, giving around 66 % accuracy. As we can see in **Figure 6**, SVM was fairly consistent, correctly classifying and misclassifying each class of music about the same percentage of the time, while KNN got a bulk of its correct classifications from class three (heavy metal). Interestingly, the Decision Tree Classification heavily favored the prediction of class three (heavy metal) regardless of the actual class of the data. This is likely what lead to its poor accuracy.

5 Summary and Conclusions

From the Yale Faces example, we saw that the SVD cannot handle translated data very well. Without the preprocessing steps required to carefully center the face, the dimension of the uncropped images under SVD quickly blew up. Not only did this make it difficult to interpret the underlying features being captured by each mode, but it drastically increased the computational work needed to manipulate this system. As we saw in the Music Classification section, if we try to apply this SVD decomposed system to a classification problem, we often run into slow runtimes or overfitting. Instead, if we invest a bit of time into preprocessing our data, we can generate a low-rank approximation. From this approximation, we can pick out only the principal coordinates that best separate the data. Rather than working with the entire dataset, we only have to work with the coordinates of our data in this reduced-dimension space. Not only will this reduce the likelihood of overfitting, but it also makes our system easier to work with computationally. This may allow us to train on larger datasets and improve the quality of our predictions.

6 Appendix A

Below is a brief summary of the MATLAB functions I used during this project and their functions.

svd(X): Calculates the Singular Value Decomposition of the given matrix X, returning U, Σ, V.

fitcecoc(training data, labels): Generates a predictive model using multi-class SVM, training it with the given data and its associated labels.

fitcknn(training data, labels): Generates a predictive model using K-Nearest Neighbors, training it with the given data and its associated labels.

fitctree(training data, labels): Generates a predictive model using Decision Tree Learning, training it with the given data and its associated labels.

fft(x): Computes the frequency content of the given vector.

pcolor(X): plots a colored representation of the given matrix.

flipud(X): Flips a matrix top to bottom to be compatible with pcolor.

7 Appendix B

7.1 Main Methods

7.1.1 Part 1: Yale Faces

```
% AMATH 482 HW 4

% Zachary McNulty

% NOTES: benefits of alignment is that we can drastically reduce the
    number of modes
% we have. SVD SUCKS with translated data so the uncropped images raise
     the supposed
% rank of the system becasue SVD cannot recognized the translated
    images as
% the same. Takes time for preprocessing.

% Convert of music into frequency space to avoid the issues with
    translation that interferes
% With SVD


%% Part 1: Yale Faces CROPPED

clear all; close all; clc;

% path to hw folder
% system('cd ~/Desktop/AMATH_482/hw/hw4')

% cropped vs uncropped shows how poor SVD does with translated images
    and
% the importance of pre-processing.
folders = dir('~/Desktop/AMATH_482/hw/hw4/input_files/CroppedYale/yale*
    ');

% MY FACES!
F_ave = zeros(32256, 38);
F = zeros(32256, 2415);

index = 1;
index2 = 1;
for j = 1:length(folders)
    path = strcat(folders(j).folder , '/', folders(j).name, '/*.pgm');
    files = dir(path);

    for k = 1:length(files)
        face = imread(strcat(files(k).folder , '/', files(k).name));

        % average all the images for each person
        % Since all the images have the same positional location of the
        % face this seems to help balance out the lighting in the
        % photograph.
        F(:, index2) = reshape(double(face), size(face, 1)*size(face,2)
            , 1);
        index2 = index2 + 1;

        if k == 1
           ave_face = double(face);
        else
            ave_face = ave_face + double(face);
```

```
            end
        end
        ave_face = ave_face ./ length(files);

        % each row is a measurement of a given pixel across many trials (i.
            e.
        % photos of people)
        F_ave(:, index) = reshape(ave_face, size(ave_face, 1)*size(ave_face
            , 2), 1);
        index = index + 1;
end

% Demean the data
mean_ave = mean(F_ave, 2);
F_ave = F_ave - mean_ave;
mean_F = mean(F,2);
F = F - mean_F;

imheight = size(face,1);
imwidth = size(face,2);

%% Calculate the Singular Value Decomposition

%[u, s, v] = svd(F_ave, 'econ');
[u2, s2, v2] = svd(F, 'econ');


%% Interpretation of U, Sigma, V

% set which group of images we are working with, averaged or not
U = u;
S = s;
V = v;
X = F_ave;
m = mean_ave;

%%
% The columns of U represent an orthogonal basis for the codomain of
    our data matrix X.
% Not just any basis, however, the BEST basis for the space where each
% successive column captures as much of the variance in the data as
% possible.
% In this case, as the codomain of our matrix is the space in which
    column of
% the matrix exists, as each column is a face in our data set the
    codomain is
% "face space" Thus, it follows that
% U is the "optimal" basis for face-space where the SVD's idea of what
    the important features
% that make up a face are will be those that capture the most variance.
% Furthermore, the columns of V then store the
% coordinates of each of our data measurements (i.e. each face) within
    this
% face space. What linear combination of these principal components
% (outputted by the code below).
num_faces = 5;
for col = 1:num_faces
    figure(2)

    subplot(1, num_faces, col)
    next_face = reshape(U(:, col), [imheight, imwidth]);
```

```
    pcolor(flipud(next_face)), shading interp, colormap(gray);
    title(strcat("Principal Component ", num2str(col)));
    set(gca,'xticklabel',[])
    set(gca,'yticklabel',[])
    set(gca, 'fontsize', 15)
end



%% Plot singular Values

figure(4)
plot(diag(S) ./ max(diag(S)), 'r.', 'markersize', 20)
%xticks(1:round(length(S)/10):length(S))
xticks(0:20:240)
xlim([0, 250])
ylim([0,1])
title('Normalized Singular Values')
ylabel('\sigma_j')
xlabel('index (j)')
set(gca, 'fontsize', 20)

%% Choosing rank r

% We will choose the r that captures at least 90% of the energy of the
% system

energy = 0;
total = sum(diag(S));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.30;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + S(r,r)/total;
end


%% rank r approximation of face space

% low-rank approximation of our faces stored in F.
X_r = U(:, 1:rank) * S(1:rank, 1:rank) * (V(:, 1:rank)');

faces_to_plot = 10;

for j = 1:faces_to_plot
figure (5)
subplot(211)
imshow(uint8(reshape(X_r(:, j) + m, imheight, imwidth)));
title(strcat("Rank ", num2str(rank), " approximation"))
set(gca, 'fontsize', 15);
subplot(212)
imshow(uint8(reshape(X(:, j) + m, imheight, imwidth)));
title('Original Image')
set(gca, 'fontsize', 15);
end


%% Part 1: Yale Faces UNCROPPED

```

```
clear all; close all; clc;

% NOTE: in this case averaging might not work as well as the previous case
% because the photos are no longer aligned.

% cropped vs uncropped shows how poor SVD does with translated images and
% the importance of pre-processing.
folders = dir('~/Desktop/AMATH_482/hw/hw4/input_files/yalefaces_uncropped/subject*');

% MY FACES!
F_ave = zeros(77760, 15);
F = zeros(77760, 165);

index = 1;
index2 = 1;
for j = 1:length(folders)
    path = strcat(folders(j).folder , '/', folders(j).name, '/subject*');
    files = dir(path);

    for k = 1:length(files)
        face = imread(strcat(files(k).folder , '/', files(k).name));

        % average all the images for each person
        % Since all the images have the same positional location of the
        % face this seems to help balance out the lighting in the
        % photograph.
        F(:, index2) = reshape(double(face), size(face, 1)*size(face,2), 1);
        index2 = index2 + 1;

        if k == 1
            ave_face = double(face);
        else
            ave_face = ave_face + double(face);
        end
    end
    ave_face = ave_face ./ length(files);

    % each row is a measurement of a given pixel across many trials (i.e.
    % photos of people)
    F_ave(:, index) = reshape(ave_face, size(ave_face, 1)*size(ave_face, 2), 1);
    index = index + 1;
end

% Demean data?
mean_ave = mean(F_ave, 2);
F_ave = F_ave - mean_ave;
mean_f = mean(F, 2);
F = F - mean(F);

imheight = size(face,1);
imwidth = size(face,2);

%% Calculate the Singular Value Decomposition

```

```
[u, s, v] = svd(F_ave, 'econ');
[u2, s2, v2] = svd(F, 'econ');


%% Interpretation of U, Sigma, V

% set which group of images we are working with, averaged or not
U = u2;
S = s2;
V = v2;
X = F;
m = mean_f;


%%
% The columns of U represent an orthogonal basis for the codomain of
    our data matrix X.
% Not just any basis, however, the BEST basis for the space where each
% successive column captures as much of the variance in the data as
% possible.
% In this case, as the codomain of our matrix is the space in which
    column of
% the matrix exists, as each column is a face in our data set the
    codomain is
% "face space" Thus, it follows that
% U is the "optimal" basis for face-space where the SVD's idea of what
    the important features
% that make up a face are will be those that capture the most variance.
% Furthermore, the columns of V then store the
% coordinates of each of our data measurements (i.e. each face) within
    this
% face space. What linear combination of these principal components
% (outputted by the code below).

num_faces = 5;
for col = 1:num_faces
   figure(2)

   subplot(1, num_faces, col)
   next_face = reshape(U(:, col), [imheight, imwidth]);
   pcolor(flipud(next_face)), shading interp, colormap(gray);
   title(strcat("Principal Component ", num2str(col)));
   set(gca,'xticklabel',[])
   set(gca,'yticklabel',[])
   set(gca, 'fontsize', 15)
end




%% Plot singular Values

figure(4)
plot(diag(S) ./ max(diag(S)), 'r.', 'markersize', 20);
ylim([0,1])
title('Normalized Singular Values')
ylabel('\sigma_j')
xlabel('index (j)')
set(gca, 'fontsize', 20);
%% Choosing rank r

```

```
% We will choose the r that captures at least 90% of the energy of the
% system

energy = 0;
total = sum(diag(S));
% how much energy we want our modes to capture.
% 75% does alright; 90% does very good.
threshold = 0.9;
r = 0;
while energy < threshold
    r = r + 1;
    energy = energy + S(r,r)/total;
end


%% rank r approximation of face space

rank = r;

% low-rank approximation of our faces stored in F.
X_r = U(:, 1:rank) * S(1:rank, 1:rank) * (V(:, 1:rank)') + m;

faces_to_plot = 10;

for j = 1:faces_to_plot
figure (5)
subplot(211)
imshow(uint8(reshape(X_r(:, j), imheight, imwidth)));
title('Low rank approximation')
subplot(212)
imshow(uint8(reshape(X(:, j) + m, imheight, imwidth)));
title("Original Image")
pause(1);
end
```

7.1.2 Part 2: Music Classification

```matlab
%% Part 2: Music Classification


%% Test case 1: Different Bands from Different Genres


% Collect Data
clear all; close all; clc;

% path to hw folder;
% set the case you want to work on here: part1,part2,part3
folders = dir('~/Desktop/AMATH_482/hw/hw4/input_files/music_files/part3/*');

% number of 5 second long samples to be taken from each song
% NOTE: takes actually twice this number of samples as half the samples are
% placed in the training set and the other half in the validation set.
samples_per_song = 15;
num_songs = 21; % total number of songs across all groups
sample_length = 5; % in seconds
skip_period = 15; % skip the first "skip period" seconds of song to avoid aampling silence at beginning of song.
%sample_rate = 2; % take every other "sample_rate"th point.

% Fs is always 44100 across all the songs we downloaded based on
% the download procedure
Fs = 44100;


% Each row is a sample
% each data sample will be the frequency content of a given song
training_data = zeros(Fs*sample_length + 1, samples_per_song * num_songs);
%training_data = zeros(262152, samples_per_song * num_songs); % spectrogram
training_labels = zeros(samples_per_song * num_songs, 1);

validation_data = zeros(Fs*sample_length + 1, samples_per_song * num_songs);
%validation_data = zeros(262152, samples_per_song * num_songs); % spectro
validation_labels = zeros(samples_per_song * num_songs, 1);

index = 1;
group_num = 1;

tic
for j = 1:length(folders)

    % skip all hidden folders within the directory
    if startsWith(folders(j).name, '.')
        continue
    end

    path = strcat(folders(j).folder , '/', folders(j).name, '/*.mp3');
    files = dir(path);

    for k = 1:length(files)
```

```
        song_path = strcat(files(k).folder, "/", files(k).name);


        start = skip_period * Fs; % skip first "skip_period" seconds of song.
        finish = inf; % sample until the end of audio file

        % Fs is the sampling rate and Y is the amplitude at each point in
        % the recording
        [Y, Fs] = audioread(song_path, [start, finish]);

        % Y is a stereo measurement (includes measurements for both left
        % and right speakers) so we average these two to get a single
        % measurement

        Y = mean(Y, 2).';
        %p8 = audioplayer(Y,Fs); playblocking(p8);


        % randomly make training data set and cross validation data set
        for s = 1:samples_per_song

            % randomly choose a sample starting point
            sample_start = randi(length(Y) - sample_length*Fs, [2,1]);

            % from the given starting point, take a sample of
            % 'sample_length' seconds and store the corresponding label
            % Take the fft of the sample to convert to frequency space

            training_data(:, index) = fft(Y(sample_start(1):sample_start(1) + sample_length * Fs)).';
            %spec = spectrogram(Y(sample_start(1):sample_start(1) + sample_length * Fs));
            %training_data(:, index) = reshape(spec, size(spec,1) * size(spec,2), 1);
            training_labels(index) = group_num; % i.e. this song came from band j

            % take another sample to be used for validation
            validation_data(:, index) = fft(Y(sample_start(2):sample_start(2) + sample_length * Fs)).';

            %spec = spectrogram(Y(sample_start(2):sample_start(2) + sample_length * Fs));
            %validation_data(:, index) = reshape(spec, size(spec,1) * size(spec,2), 1);
            validation_labels(index) = group_num;
            index = index + 1;

%             if s == 1 && k == 1
%                 figure(10)
%                 t = 0:1/Fs:5;
%                 subplot(3, 1, group_num);
%                 plot(t, Y(sample_start(1):sample_start(1) + sample_length * Fs), 'r');
%
%                 xlim([0,5])
%
%                 if group_num == 1
```

```
%                         title('AC DC')
%                     elseif group_num ==2
%                      title('Beethoven')
%                     elseif group_num == 3
%                         xlabel('Time (s)')
%                         title('Frank Sinatra')
%                     end
%                     ylabel('Amplitude')
%                     set(gca, 'fontsize', 15);
%                 end

          end

      end

      group_num = group_num + 1;

end
toc


%% Find Modes which are best at separating groups.

% Extract the individual groups (i.e. bands in this case) from the training data
group1 = training_data(:, training_labels == 1);
group2 = training_data(:, training_labels == 2);
group3 = training_data(:, training_labels == 3);

% SVD the entire dataset to find principal components
% of "music space"
mean_td = mean(training_data, 1);
[u, s, v] = svd(training_data - mean_td, 'econ'); % mean subtract

% Plot singular values: which are relevant?

singular_values = diag(s) / max(diag(s));
plot(singular_values, 'r.', 'markersize', 20)
title('Normalized Singular Values')
ylabel('\sigma_j')
xlabel('index j')

% Find the modes that best separate the data by projecting each of our
% groups onto the individual components. Note that each column of V gives
% the coordinates of each data measurement (row in X = training_data) onto
% the corresponding principal component (column in U). So entry (i,j) of V
% gives the weighting/"importance" of principal component


%% Low rank approximation

% Calculate the low-rank approximation that captures at least 90% of the
% energy of the system.
energy_threshold = 0.90;
rank = find(cumsum(singular_values / sum(singular_values)) >= energy_threshold, 1);

```

```
X_r = u(:, 1:rank) * s(1:rank, 1:rank) * (v(:, 1:rank)') + mean_td;

%% Test Quality of reconstruction

% test = real(ifft(X_r(:, 1))); % all complex parts are zero and
    interfere with playing music
% p8 = audioplayer(test, Fs);
% playblocking(p8);

%% Train a SVM model
clc;

% SVM Cannot handle complex numbers so we simply take the absolute
    value
% To convert the training data to amplitude (all real).

% In this case, as our data measurements are stored as columns, the
    columns of
% U store the fundamental structure/modes of our system (they form a
    basis
% for our domain : frequency / music space) while the columns of SV'
    give the
% coordinates of each data measurement (i.e. each song) within this
% frequency space. Thus, these coordinates define the underlying
    structure
% of our system and we can use them to classify.

% Specifically, we will use V* as our "coordinates" and project future
% data measurements onto the columns of U*S

vstar = v';
xtrain = [real(vstar(1:rank, :)) ; imag(vstar(1:rank, :))].';

%SVM model
Mdl_svm = fitcecoc(xtrain, training_labels);

% K nearest neighbors model
Mdl_knn = fitcknn(xtrain, training_labels);

% Naive Bayes Model
%Mdl_nb = fitcnb(xtrain, training_labels);

% Decision Tree Classification model
Mdl_tree = fitctree(xtrain, training_labels);


% Computes the classification Error rate in classification for our
    training
% data set using the classification conditions defined by our model.
error_rate_training_svm = resubLoss(Mdl_svm)
error_rate_training_knn = resubLoss(Mdl_knn)
%error_rate_training_nb = resubLoss(Mdl_nb)
error_rate_training_tree = resubLoss(Mdl_tree)

%% Test how good the model is at predicting labels


% since our model was trained on the coordinates given by V which are
    the coordinates
% of our data with respect to the basis US (for frequency space), we
    have to
```

```
% to get the coordinates of our validation data with respect to this
% basis: US * coordinates_in_US = data. Since U is unitary, this yields
% simply: coordinates_in_US = S^-1 U' * data
Sinv = diag(1 / diag(s));
coordinates_in_US = (u')*validation_data;
xtest = [real(coordinates_in_US(1:rank, :)) ; imag(coordinates_in_US(1:
    rank, :))].';

% we only used the first r = rank coordinates to determine our
% classification so we again do that.
predictions_svm = predict(Mdl_svm, xtest);
predictions_knn = predict(Mdl_knn, xtest);
%predictions_nb = predict(Mdl_nb, xtest);
predictions_tree = predict(Mdl_tree, xtest);
accuracy_svm = sum(predictions_svm == validation_labels) / length(
    validation_labels)
accuracy_knn = sum(predictions_knn == validation_labels) / length(
    validation_labels)
%accuracy_nb = sum(predictions_nb == validation_labels) / length(
    validation_labels)
accuracy_tree = sum(predictions_tree == validation_labels) / length(
    validation_labels)




%% Find which kinds of misclassifications were common.

figure(10)
subplot(131)
classification_heatmap(validation_labels, predictions_svm);
title('SVM Classifications')
subplot(132)
classification_heatmap(validation_labels, predictions_knn);
title('KNN Classifications')
subplot(133)
classification_heatmap(validation_labels, predictions_tree);
title('Decision Tree Classifications')
```

7.2 Helper Methods

```
function x = classification_heatmap(actual, predictions)
% returns a heatmap where columns are "classified as" and rows are "actual"
x = zeros(3,3);
for j = 1:3
    for k = 1:3
        x(j, k) = sum((actual == j) .* (predictions == k));
    end
end

heatmap(x);
ylabel('Actual Class')
xlabel('Predicted Class')
set(gca, 'fontsize', 20);
colorbar( gca , 'off' )

end
```