Principal Component Analysis of Faces and Classification of Musical Genres and Artists with Linear and Quaratic Discriminant Analysis

Trent Yarosevich[1]

Abstract
This assignment is comprised of two parts. In Part 1, we use Principal Component Analysis (PCA) to examine a large set of images of human faces in an attempt to assess the effectiveness of rank reduction and study the strongest components of the human face. In Part 2, we use a training set of 5-second samples to evaluate the efficacy of several classifying tools based on the singular value decomposition (SVD), namely Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA). Non SVD based classifiers are also used as a point of comparison. This approach for music classification is done in three parts, which attempt to distinguish: different artists from different genres; different artists from within the same genre; and different genres from one another.
https://github.com/tyarosevich/AMATH_582/blob/master/HW4/yarosevich_582_hw4_writeup.pdf

[1] *Department of Applied Mathematics, University of Washington, Seattle*

Contents

1. Introduction and Overview

One of the myriad uses of the SVD lies in machine learning algorithms involved with classification of data, stemming from the SVD's ability to provide the most accurate (in an L-2 sense) reduced rank representations of data, allowing for an efficient statistical comparison of high-dimensional systems. In the following experiment I will use this strategy to attempt to both analyze human faces and classify music in three parts with progressively increasing degrees of subjective similarity between samples.

In Part 1, I will study two datasets of several thousand cropped images, and several hundred uncropped images, of human faces. Using the SVD I will then study the principal components of this data set and discuss the 'average' face as well as the impact of using uncropped images.

In Part 2, I will attempt to classify music in three sections. First, I compare three artists from genres that most subjective human tastes would say are very different: 90's Hip-Hop icon GZA, iconic thrash-metal band Metallica, and avante-garde progressive electronic artist Squarepusher. Second, I will compare three progressive electronic artists, Squarepusher, Aphex Twin, and Plaid, who all have their origins at the iconic experimental electronic label, Warp Records. Finally, I will compare not only three genres, but three sub-genres of metal: thrash metal, progressive metal, and death metal. These samples will then be randomly divided into training and testing sets and the results evaluated.

2. Theoretical Background

2.1 Data Processing

No special pre-processing was required for the first part of this assignment. The images are simply flattened and then collected in a matrix for analysis. Part 2, on the other hand, does require a few processing steps that should be noted. Once the music is pre-processed by converting to .wav format, mono sound, and being stripped of all metadata, a spectrogram was extracted from the data. This was accomplished with scipy's .signal library, which includes a spectrogram method. This method creates a spectrogram with a sequenced of windowed fast-fourier transforms (FFT), which allows for an approximation of the frequency-domain's development over time. The window used was the default tapered-cosine window (Tukey).

The purpose of this transform is to create a frequency context for the analysis of the signals. In the spatial domain, a song can vary wildly between 5-second samples. However, in the frequency domain the common instrumentation, voices, production artifacts, and other idiosyncracies that stretch across the entire song, and potentially the entire artist or genre, are potentially represented in each sample. Additionally, the spectrogram also allows for substantial reduction in data size, since it is quite easy to eliminate frequency ranges that contain very little information - in this experiment, roughly 5,000 hz were used since the majority of the energy occurs in this frequency range, and experimentation showed this truncation had little effect on accuracy (and in some cases improved it).

2.2 The SVD

The singular value decomposition is widely discussed in the educational literature, and so I will not go into detail here beyond discussing the salient features relevant to this analysis. Intuitively, the SVD decomposes a matrix into three matrices that rotate, stretch, and then again rotate any matrix to which they are applied. Crucially, these rotation matrices are always unitary[1], and thus for a given rotation matrix R we have $R^T = R^{-1}$. Furthermore, the 'stretch' matrix is simply applying a scalar multiple to each dimension of the target matrix, and is thus diagonal. The SVD is thus written as,

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

where $\hat{U}$ and V^* 'rotate' and are unitary, and Σ 'stretches' and is diagonal. In this case, we are referring to the 'reduced' SVD, which differs from the 'Full' SVD in two ways: first, it omits the silent columns of U that actually make it a unitary matrix, which is why it is written as $\hat{U}$; second it omits the additional zero singular values in Σ that correct for these silent columns in the final product. Generally speaking, the reduced SVD is used unless the unitary property is expressly required.

There are numerous theorems associated with the SVD, but I will only briefly mention the ones that inform my algorithm. Note that the proofs are summarized from more complete statements[2].

Theorem 1: *Every Matrix A has a singular value decomposition*

Theorem 2: *A is the sum of r rank one matrices,* $A = \sum_{j=1}^{r} \sigma_j u_j v_j^*$.

Theorem 3: *For any N such that* $0 \leq N \leq r$ *we define the partial sum* $A_N = \sum_{j=1}^{N} \sigma_j u_j v_j^*$, *and if* $N = min\{m,n\}$, *define* $\sigma_{N+1} = 0$, *then* $\|A - A_N\| = \sigma_{N+1}$.

While the proof of existence is obviously important, it is **Theorem 3** and its interpretation that is perhaps most important here. In short, this theorem guarantees that any reduced rank representation of a matrix A using the SVD is *guaranteed* to be the best representation possible in an L-2 norm (or Frobenius) sense. This result underscores the power and viability of PCA.

2.3 Principal Component Analysis (PCA)

A useful way to introduce PCA is to consider the variance and covariance among a set of measurements (precisely what this investigation will do). Suppose we have a matrix of data, in which each row represents a measurement (or each pair of rows, etc.) vector:

$$X = \begin{bmatrix} x_a \\ x_b \\ x_c \end{bmatrix} \tag{2}$$

The variance and covariance of these vectors are given by, e.g. $\sigma_a^2 = \frac{1}{1-n} x_a x_a^T$ and $\sigma_a^2 b = \frac{1}{1-n} x_a x_b^T$. Following from this we can represent an entire matrix of the variance and covariance terms called the Covariance Matrix, given by

$$C_x = \frac{1}{1-n} XX^T \tag{3}$$

in which the diagonal terms represent the variance, and the off-diagonal terms represent all possible covariance combinations between measurement vectors.

The connection with the SVD lies precisely with this covariance matrix. The SVD of X *diagonalizes* the covariance matrix by representing X in a new set of orthogonal bases, $\hat{U}$ and V^*, which remove all redundancy, that is to say covariance, from the matrix as well as ordering the variance values on the diagonal of Σ. This is easily illustrated by calculating the covariance matrix of X projected onto the unitary transformation U^* from the Full SVD, $Y = U^*X$:

$$C_Y = YY^T = \frac{1}{n-1}(U^*X)(U*X)^T = \frac{1}{n-1}\Sigma^2 \tag{4}$$

Thus since, by **Theorem 1** the SVD is guaranteed to exist, this removal of redundancy is always possible.

Once this redundancy is removed, several powerful analytical tools are opened up. For one, we can use this PCA method

[1] "Rotation Matrix." Wikipedia, Wikimedia Foundation, 20 Feb. 2020, en.wikipedia.org/wiki/Rotation_matrix.

[2] Kutz, Jose Nathan. Data-Driven Modeling and Scientific Computation. Methods for Complex Systems and Big Data. Oxford University Press, 2013, page 392, 399.

to examine the how much of the total energy or variance is captured by each mode by examining their relative weights, where σ_e is a vector representing the proportion of energy contained in each value, and σ_v is the total variance:

$$\begin{aligned}\sigma_e &= \frac{\sigma_j}{\sum_{j=1}^{m}\sigma_j} \\ \sigma_v &= \frac{\sigma_j^2}{\sum_{j=1}^{m}\sigma_j^2}\end{aligned} \tag{5}$$

We can then use these weights alongside an examination of the bases to make observations about the dynamics of the original data. For example, we can project the orignal data in the direction of the orthogonal spatial modes $\hat{U}$ and make observations about what the behavior of the first, dominant mode is.

2.4 Principal Orthogonal Modes (POD) and Linear Discriminant Analysis (LDA)

Another tool the SVD opens up to us is POD. With this approach, the orthogonal bases themselves are studied in order to try to understand what the data does when projected onto these bases - bases which eliminate redundancy (co-variance). This then forms the basis of LDA, which is at the core of this experiment.

In Part 1, the principal modes can be plotted to see what the 'average' face looks like (the column of U associated with the first, or largest, singular value) as well as what the 'face' associated with the other singular values looks like.

In part 2, another aspect of POD is used, and this is what leads to LDA. By examining the rows of V^*, we can study how strong these modes are in particular samples of music by looking at a particular sample's projection onto ΣV^*. The hope is then that different artists or genres, for example, will have common features in a particular mode that can then be used to distinguish them from one another and thereby classify them.

Once a suitable mode is found that distinguishes labeled classes of data, linear discriminant analysis is used to project the data onto a subspace that both (a) maximizes the distance between the labeled classes of data, (b) minimizes the intra-class distance between points within a single labeled class. This is an optimization problem in which the between and within class matrices are given by:

$$\begin{aligned}S_B &= (\mu_2-\mu_1)(\mu_2-\mu_1)^T \\ S_W &= \sum_{j=1}^{2}\sum_{x}(x-\mu_j)(x-\mu_j)^T\end{aligned} \tag{6}$$

These values are then used for the optimzation problem:

$$w = argmax_w \frac{w^T S_B w}{w^T S_W w} \tag{7}$$

Finally, this value is used to solved the eigenvalue problem $S_B w = \lambda S_W w$ in which λ gives the mode of interest and the associated eigenvector is the projection basis. In the subsequent analysis, this method will be handled with built-in methods of the sklearn library.

3. Algorithm Implementation and Development

3.1 Part 1 - PCA and Images of Faces

For this section, Matlab was used and the analysis was conducted with the following steps.

3.1.1 1.1: Data Pre-Processing

The directory containing the images was iterated over and each image was read in from file, flattened into a column, and stored in a master matrix. This was repeated for the uncropped images, and then both matrices were subjected to an SVD and exported to file.

3.1.2 1.2: Plotting

The bulk of Part 1's result was the plots themselves, as they are the subject of discussion. First, the relative energy contained in each singular value was plotted, on both a standard and partial log (y-axis) scale. Next the four dominant modes were plotted by reshaping the associated columns of U back into matrices of the same resolution as the original file, which were then converted to grayscale values using matlab's built-in `mat2gray()` function. This was then repeated for the uncropped images.

3.2 Part 2 - Music Classification

3.2.1 2.1 - Data Pre-Processing

Data processing was more elaborate for this section, and was done using python in order to streamline the code and access powerful libraries to do the mathematical heavy-lifting.

First, the music samples were organized in a file tree. Each folder was named using a numerical order which insured iteration would proceed in a predictable manner. The `os.walk()` function was ten used to iterate through these folders and at each step the following occurred:
(1) - Ten randomly generated 5-second segments were defined.
(2) - A spectrogram of each of these was taken, and the spectrogram values were flattened into a column using `np.reshape()`. This vector was then trimmed of the last roughly 80% of values in order to remove frequencies above roughly 5 kHz, since the spectrogram values in this range become increasingly smaller, and doing so had no impact on accuracy (removing at least 5 kHz actually improved it). Figure 1 gives an example of a typical spectrogram frequency range. (3) - The folder from which each sample is taken is stored, and then these values are appended to holder variables.

The function to get these samples is then used to collect a large data matrix containing all 900 samples used for the entire project. The labels returned from this matrix are then converted to strings representing each artist or genre using a dictionary, and a matrix is created for each part using the

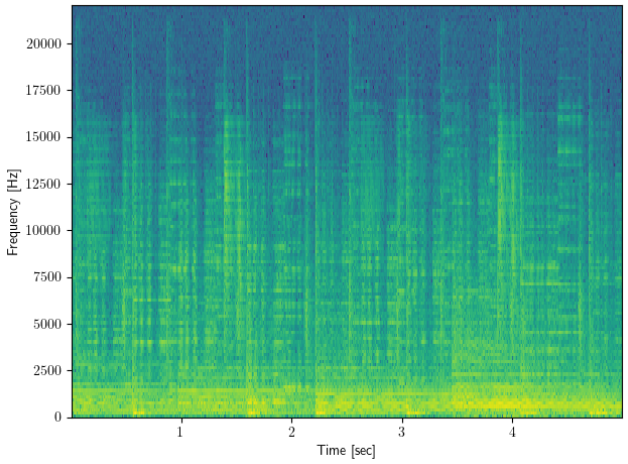

Figure 1. A spectrogram of "OH" by the group Plaid

appropriate data and labels.

3.2.2 2.2 - Plotting POD modes

For each section, an SVD is taken of the data and the strength of each sample is projected onto the first three modes and plotted iteratevly for each artist or genre, ostenibly to demonstrate the viability of finding a linear discrminant. The first 50 singular values are also plotted. In all cases the standard pyplot library is used.

3.2.3 2.3 - Classification

Finally, each section's data is classified using LDA, QDA, support vector classification (SVC), and a naive Bayes classifier, the latter two being present simply for comparison to the more central LDA and QDA. These classifications are done using built-in methods in the `scikit learn` library and all proceed similarly.

First the `train_test_split ()` method is used to divide the overall data set and labels into training and testing sets at an 80-20 ratio. These are randomly assigned by the method. The resulting training and testing sets are then scaled by `sc. fit_transform ()` and `sc.transform()` respectively. Finally, the classifier object is declared and is fit to the training data by passing it the data and labels, then used to score the test data.

4. Computational Results

4.1 Part 1

Figure 3 and Figure **??** show the relative weights of each singular value for the cropped image set and uncropped set respectively, while Figure 4 in turn shows the cumulative energy contained in the modes for the cropped data. Figure 6 shows several rank reductions of an arbitrarily chosen cropped image.

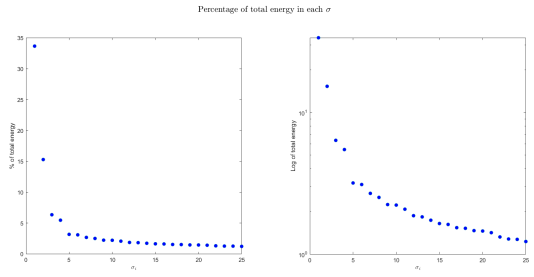

Figure 2. The relative energy per mode of the cropped face-space

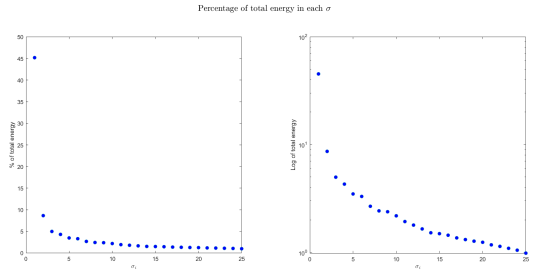

Figure 3. The relative energy per mode of the uncropped face-space

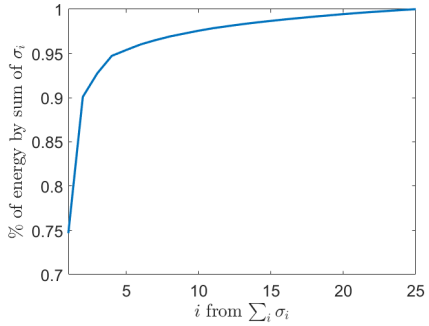

Figure 4. The cumulative energy of each rank, cropped faces.

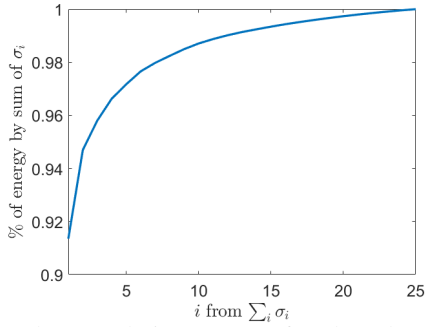

Figure 5. The cumulative energy of each rank, uncropped faces.

4.2 Part 2 - Music Classifictaion

The results for each part of the experiment are tabulated in Table 1. Singular values were similar across all parts, with dominance by one singular value, and a long tail of decreasing values as shown in Figure **??**, with accompanying projection on the the dominant modes of the first ten samples of each

Figure 6. A comparison of r=25, 50, 100 rank and full rank

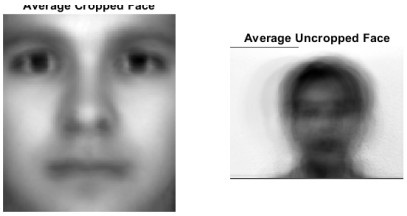

Figure 7. Cropped and Uncropped Average Faces

genre in Figure **??**.

5. Summary and Conclusions

5.1 Part 1 - Faces

The analysis showed that a striking amount of energy is contained in just a few modes, with very nearly 100% of the energy contained in the first 25. We can therefore make a case that the face-space is rank 25 - a substantial reduction from the rank 168 of the image file. Interestingly, however, to the human eye this rank 25 representation does not bear a tremendous resemblance to the original file, as shown in Figure 6. The uncropped images, by comparison showed both some expected and unexpected results. On the one hand, the faces associaed with the uncropped principle orthogonal modes are distorted, as we would expect. By observing that the U columns are the 'faces' associated with a given mode, we can use Figure 7 to compare the 'average' faces of cropped and uncroppe data sets. Despite the obvious difference, however, I was surprised to find that in the case of the cropped faces, the face space is still dominated by a similar number of modes, as shown in Figure **??**.

While further experimentation would seem to be in order, it seems clear that while a majority of 'energy' is contained in just the first 25 modes, it would seem that the relatively 'small' modes can still play a large role in human facial recognition. Furthermore, while cropping has a significant impact on average faces, it does not seem to have a huge impact on the spread of singular values - that is to say, in an ideal basis, the types of structures in the data seem to remain consistent, even if the projection onto these bases produced blurred images due to the 'noise' of inconsistent cropping.

	Part 1	Part 2	Part 3
LDA	.65	.43	.63
QDA	.3	.28	.42
SVM	.65	.45	.52
NB	.42	.43	.53
KNN	.32	.61	.45

Table 1. % of correctly classified test data

5.2 Part 2 - Music Classification

While many of the algorithms, including LDA, produced far better results than guessing, the classification of music was not very good. I would speculate that part of this has to do with how humans distinguish sounds versus a mathematical process. For example, timbre, which humans use to distinguish music with ease, is mathematically very insignificant compared other frequency components, and indeed in many cases might be smaller than signal noise. Case in point, Figure **??** shows that there is no clear discriminant line in the first three POD projections for Part 2.3. In general, while I expected a decreasing degree of success when moving from Parts 1 to Part 3, it was quite surprising to see that Part 2 had a lower accuracy than Part 3. This would seem to suggest that the statistical distinction between genres and artists in this type of analysis does not conform to our own intuition.

In closing it is worth noting that, despite the somewhat low success rate, the algorithm's ability to classify metal subgenres with 63% accuracy was impressive, considering the variety contained in progressie sub-genres. For example, a human unfamiliar with the artists involved in the metal classification would likely struggle to classify them after listening to many samples due to the disparity of styles present in a single song from, for example, Between the Buried And Me, who routinely insert non-metal genre segments within their songs. Overall, the results were promising considering this was a first pass at classification, and refinement of the methods could still be possible.

,,,,,,,,

,,,,,,,,

Appendix 1

Matlab Functions:

dir(name): Returns a structure array containing the data attributes of the argument, which is a folder filepath or folder name.

fullfile (name): Returns the full file path of the passed folder, which in this case must be present in the workplace directory.

U, S, V = **svd**(A): Returns the singular value decomposition of matrix A. The option '0' can also be passed to produce a reduced SVD.

imread(A): Reads in the image A, where A is the file path passsed as a string.

imshow(A): Displays the image using the variable A, which is a matrix of RGB or grayscale values.

mat2gray(M): Rescales the matrix M to appropriate grayscale saturation values.

reshape(A, m, n: Reshapes the matrix A to the dimensions mxn. **cumsum**(v): Returns a vector containing the cumulative sum of the argument. For example, if [a b c] is passed, it returns [a, a + b, a + b + c].

Python Functions:

os.walk(**dir**): Returns a string containing the root directory, a list containing the sub-directories as strings, and a list containing the files in the subdirectories as strings. Extremely useful for iterating over folder contents.

freq, t, spec = signal.spectrogram(fs, sample): Returns y (power spectrum), time, and spectrogram components of the sample. The sampling rate fs must also be passed to the method.

np.**reshape**(A, −1): Flattens the matrix A into a column.

values = [dictionary [k] **for** k in list]: A useful list comprehension that returns a list of values associated with a list of keys by referencing a dictionary.

sklearn: This library is too extensive to discuss here, and discussing even one of the methods used would require an excess of space. Instead, I provide urls to the relevant documentation:
https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscrim
html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC.
score
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#
sklearn.naive_bayes.GaussianNB.score
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.
html
https://scikit-learn.org/0.16/modules/generated/sklearn.qda.QDA.html

Appendix 2

```
%% HW 4 Part 1
% Import and store the image files.
% Note the cropped images are 192x168
clc; clear all; close all

D = 'CroppedYale';
S = dir(fullfile(D));

% This matrix stores every image, from every sub-folder, in a large
```

```
% uint8 type matrix.
cropped_master_mat = [];

% Iterate through the contents of the folder, and one subfolder down.
% Please note that I obtained this code from Matlab's official forums as I
% do not have time to wade through the quirks of how matlab handles chars,
% workspace, and folders.

% The directory containing the subfolders
D = 'C:\Users\tyran\Dropbox\School Stuff\Winter 2020\AMATH_582\HW4\CroppedYale';

 %Makes a list of the directory of the folder containing the sub-folders.
S = dir(fullfile(D,'*'));

% removes the up/down dirs and makes a list of the folder names.
N = setdiff({S([S.isdir]).name},{'.','..'});

for i = 1:numel(N)

    T = dir(fullfile(D,N{i},'*')); %sub-folder directory
    C = {T(~[T.isdir]).name}; % files in subfolder.
    for j = 1:numel(C)
        F = fullfile(D,N{i},C{j});
        % do whatever with file F.
        I = imread(F);
        I_col = I(:) - mean(I(:));
        cropped_master_mat = [cropped_master_mat I_col];

    end
end

%% Save the file

save('cropped_matrix.mat', 'cropped_master_mat')
writematrix(cropped_master_mat, 'cropped_matrix.csv')

%% Take an SVD and export
A = double(cropped_master_mat);
[U, S, V] = svd(A, 'econ');
save('centered_cropped_svd', 'U', 'S', 'V')


%% Uncropped face
% Import and arrange the uncropped faces and save the matrix file locally.
% Note these files are 243x320 pixels.
clc; clear all; close all;

D = 'C:\Users\tyran\Dropbox\School Stuff\Winter 2020\AMATH_582\HW4\yalefaces';
uncropped_master_mat = [];
T = dir(fullfile(D,'*')); %sub-folder directory
C = {T(~[T.isdir]).name}; % files in subfolder.
for j = 1:numel(C)
    F = fullfile(D,C{j});
    % do whatever with file F.
    I = imread(F);
    I_col = I(:);
    uncropped_master_mat = [uncropped_master_mat I_col];

end


%% Save the matrix

save('uncropped_matrix.mat', 'uncropped_master_mat')
writematrix(uncropped_master_mat, 'uncropped_matrix.csv')

%% Take an svd and export

A = double(uncropped_master_mat);
[U, S, V] = svd(A, 'econ');
```

```
save('uncropped_svd', 'U', 'S', 'V')

%% HW 4 Part 1 Processing and Analysis
clc; clear all; close all

%% Cropped Faces

load('centered_cropped_svd.mat')

%% Sigma plots
sig = diag(S);
sig = sig(1:25);
% energy and log energy plots
figure(1)
subplot(1,2,1)
plot((sig / sum(sig)) * 100,'o', 'MarkerFaceColor', 'b')
xlabel('$\sigma_i$', 'Interpreter', 'latex')
ylabel('% of total energy')
set(gca, 'Fontsize', [10])
axis('square')
subplot(1,2,2)
semilogy((sig / sum(sig)) * 100,'o', 'MarkerFaceColor', 'b')
xlabel('$\sigma_i$', 'Interpreter', 'latex')
ylabel('Log of total energy')
set(gca, 'Fontsize', [10])
axis('square')
sgtitle('Percentage of total energy in each $\sigma$', 'interpreter', 'latex')

%% Cumulative energy
x_s = 1:25;
s_cum = cumsum(diag(sig) / sum(diag(sig)));


plot(x_s,s_cum, 'linewidth', 2)
xlabel('$i$ from $\sum_i\sigma_i$', 'Interpreter', 'latex')
ylabel('\% of energy by sum of $\sigma_i$', 'Interpreter', 'latex')
set(gca, 'Fontsize', [15])
xlim([1 25])
%% Plot the four dominant modes in grayscale

for k = 1:4
    subplot(2,2,k)
    im = mat2gray(-reshape(U(:,k), 192, 168) );
    imshow(im)
    title('POD Mode 1')
end
%%
cropped_average = mat2gray(-reshape(U(:,1), 192, 168) );
save('cropped_average.mat', 'cropped_average')
imshow(cropped_average)

%% Rank reduction comparison
close all
full_rank = U*S*V';
test_full = mat2gray(reshape( full_rank(:,27), 192, 168));
partial_rank = U(:,1:25)*S(1:25, 1:25)*V(:, 1:25)';
test_partial = mat2gray(reshape( partial_rank(:,27), 192, 168));
partial_rank2 = U(:,1:50)*S(1:50, 1:50)*V(:, 1:50)';
test_partial2 = mat2gray(reshape( partial_rank2(:,27), 192, 168));
partial_rank3 = U(:,1:100)*S(1:100, 1:100)*V(:, 1:100)';
test_partial3 = mat2gray(reshape( partial_rank3(:,27), 192, 168));


subplot(2,2,4)
imshow(test_full)
title('A full rank image')
subplot(2,2,1)
imshow(test_partial)
title('A rank 25 image')
subplot(2,2,2)
```

```
imshow(test_partial2)
title('A rank 50')
subplot(2,2,3)
imshow(test_partial3)
title('A rank 100')


%% Uncropped Faces
clc; clear all; close all;
load('uncropped_svd.mat')

%% Sigma plots
sig = diag(S);
sig = sig(1:25);
% energy and log energy plots
figure(1)
subplot(1,2,1)
plot((sig / sum(sig)) * 100,'o', 'MarkerFaceColor', 'b')
xlabel('$\sigma_i$', 'Interpreter', 'latex')
ylabel('% of total energy')
set(gca, 'Fontsize', [10])
axis('square')
subplot(1,2,2)
semilogy((sig / sum(sig)) * 100,'o', 'MarkerFaceColor', 'b')
xlabel('$\sigma_i$', 'Interpreter', 'latex')
ylabel('Log of total energy')
set(gca, 'Fontsize', [10])
axis('square')
sgtitle('Percentage of total energy in each $\sigma$', 'interpreter', 'latex')
%% Cumulative energy
x_s = 1:25;
s_cum = cumsum(diag(sig) / sum(diag(sig)));


plot(x_s,s_cum, 'linewidth', 2)
xlabel('$i$ from $\sum_i\sigma_i$', 'Interpreter', 'latex')
ylabel('\% of energy by sum of $\sigma_i$', 'Interpreter', 'latex')
set(gca, 'Fontsize', [15])
xlim([1 25])

%% Plot the four dominant modes in grayscale

for k = 1:4
    subplot(2,2,k)
    im = mat2gray(-reshape(U(:,k), 243, 320) );
    imshow(im)
    title('POD Mode 1')
end

%% Compare average face cropped and uncropped

subplot(1,2,2)
im = mat2gray(-reshape(U(:,1), 243, 320));
imshow(im)
title('Average Uncropped Face')
subplot(1,2,1)
imshow(cropped_average)
title('Average Cropped Face')
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy import signal
import os
import random

    #%% #%% Defines a function to get samples from my folder with processed .wav files

# This dictionary gives the folder names as keys to the artists inside them.
conversion = {'1_1': 'gza', '1_2': 'metallica', '1_3': 'squarepusher', '2_1': 'squarepusher', '2_2': '
```

```
    aphex', '2_3':'plaid', '3_1':'btbam', '3_2': 'opeth', '3_3': 'tess'}

# A sample that returns 900 samples and the associated artist folder name.
def get_900samples():
    sample_size = 220500
    # Laptop
    rootdir = 'C:/wav_output/582_hw4_conversion_folder'

    # Home Desktop
    # rootdir = 'C:/582_hw4_conversion_folder/wav_output/582_hw4_conversion_folder'

    master_wav = np.zeros((35000, 900), int)
    index = 0
    sample_order = []
    song_order = []

    # Iterates through the base folder with os.walk, which goes through each subfolder, with
    # iterate variables for the files, the subdirectories and the directories.
    for subdir, dirs, files in os.walk(rootdir):

        # Iterates through each file.
        for file in files:
            # Each filepath, should we want to save it for any reason.
            file_path = os.path.join(subdir, file)
            # Reads in the .wav file
            fs, data = wavfile.read(file_path)
            # A holder for the ten samples we take from each song.
            ten_samples = np.zeros((35000, 10), int)
            # An iteration to take the 10 samples.
            for k in np.linspace(0, 9, 10, dtype = int):
                # Saves the current folder in order to confirm order of samples
                sample_order.append(os.path.basename(os.path.normpath(subdir)))
                # A random index to start the sample from.
                i = random.choice( range( len( data[2*sample_size:-2*sample_size] ) ) )
                sample = data[i:i+sample_size]
                # Takes a spectrogram of the file and reshapes it into a vector.
                frequencies, times, spectrogram = signal.spectrogram(sample, fs)
                col = np.reshape(spectrogram, -1)
                ten_samples[:,k] = col[0:35000]
            # Removes the top 5,000 hz
            master_wav[:, index*10:index*10+10] = ten_samples
            index += 1
    return master_wav, sample_order;

#%% Produce the data matrix
master_data, folders  = get_900samples()
# Returns a list of values associated with a list of keys.
master_labels = [conversion[q] for q in folders]

#%%
# Perform a reduced SVD of the data for Part 1 and plot the singular values on a standard and semi-log
    axis.
# Variables to change to re-run the code for parts 1, 2, 3.
start = 600
stop = 900
A1 = master_data[0:,start:stop]

U, S, V = np.linalg.svd(A1, full_matrices=False)
x = np.linspace(1, 50, 50)

# Plots the first 50 singular values.
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Relative Comparison of Singular Values')
ax1.plot(x, S[0:50], 'r-o')
ax1.set_xlabel('$\sigma_j$')
ax1.set_ylabel('$\sigma$ value')

ax2.semilogy(x, S[0:50], 'k-o', )
plt.rc('text', usetex=True)
ax2.set_xlabel('$\sigma_j$')
```

```
ax2.set_ylabel('log of $\sigma$ value')
plt.show()

#%%
# Plot projection onto dominant POD to look for discrimination
# subtitles = ['gza', 'gza', 'gza', 'metallica', 'metallica', 'metallica','squarepusher', 'squarepusher','squarepusher']
# subtitles = ['squarepusher', 'squarepusher','squarepusher', 'aphex twin', 'aphex twin', 'aphex twin', 'plaid', 'plaid', 'plaid']
subtitles = ['thrash', 'thrash', 'thrash', 'prog', 'prog', 'prog', 'death', 'death', 'death']
plt.figure(1)
fig1, axs1 = plt.subplots(3,3)
x = np.linspace(0, 5, 10)
ymin = -0.1
ymax = 0.1
# A highly compacted iteration to plot 9 subplots.
for i, ax in enumerate(np.reshape(axs1.T, -1)):
    idx = i//3
    ax.plot(x, V[i - 3*idx , idx*100:idx*100+10])
    ax.set_title(subtitles[i])
    ax.set(xlim = (0,5), ylim = (ymin, ymax))
plt.show()


#%% Test with sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

data = master_data[:, start:stop].T
classes = master_labels[start:stop]

# Splits the data into a training set and randomized test set with accompanying labels
X_train, X_test, y_train, y_test = train_test_split(data, classes, test_size=0.2)

# Scales the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Perform the LDA with one component
lda = LDA(n_components=2)
X_train = lda.fit(X_train, y_train)
result = lda.score(X_test,y_test)
print('Score: ' + str(result))
# X_test = lda.transform(X_test)

#%% Test with SVM
from sklearn import svm
# Splits the data into a training set and randomized test set with accompanying labels
X_train, X_test, y_train, y_test = train_test_split(data, classes, test_size=0.2, random_state=0)

# Scales the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

clf = svm.SVC(kernel)
clf.fit(X_train, y_train)
print('Accuracy of SVM classifier on training set: {:.2f}'
      .format(clf.score(X_train, y_train)))
print('Accuracy of SVM classifier on test set: {:.2f}'
      .format(clf.score(X_test, y_test)))

#%% Test with QDA
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA

# Splits the data into a training set and randomized test set with accompanying labels
X_train, X_test, y_train, y_test = train_test_split(data, classes, test_size=0.2)

```

```
148 # Scales the data
149 sc = StandardScaler()
150 X_train = sc.fit_transform(X_train)
151 X_test = sc.transform(X_test)
152
153 clf = QDA()
154 clf.fit(X_train, y_train)
155 print('Accuracy of QDA classifier on training set: {:.2f}'
156       .format(clf.score(X_train, y_train)))
157 print('Accuracy of QDA classifier on test set: {:.2f}'
158       .format(clf.score(X_test, y_test)))
159
160 #%% Test with Naive Bayes
161 from sklearn.naive_bayes import GaussianNB as GNB
162 # Splits the data into a training set and randomized test set with accompanying labels
163 X_train, X_test, y_train, y_test = train_test_split(data, classes, test_size=0.2)
164
165 # Scales the data
166 sc = StandardScaler()
167 X_train = sc.fit_transform(X_train)
168 X_test = sc.transform(X_test)
169
170 clf = GNB()
171 clf.fit(X_train, y_train)
172 print('Accuracy of Naive Bayes classifier on training set: {:.2f}'
173       .format(clf.score(X_train, y_train)))
174 print('Accuracy of Naive Bayes classifier on test set: {:.2f}'
175       .format(clf.score(X_test, y_test)))
176 #%% KNN
177 from sklearn.neighbors import KNeighborsClassifier
178 knn = KNeighborsClassifier()
179 knn.fit(X_train, y_train)
180 print('Accuracy of K-NN classifier on training set: {:.2f}'
181       .format(knn.score(X_train, y_train)))
182 print('Accuracy of K-NN classifier on test set: {:.2f}'
183       .format(knn.score(X_test, y_test)))
```