Illustration of Singular Value Decomposition with Yale Faces Dataset and Music Classifiers

Arvindh Sharma *

March 09, 2020

Abstract

Singular Value Decomposition (SVD) was used to identify key features from images in the Yale database. The decomposition was used to estimate the rank of the data matrices for cropped and uncropped image sets. The features were visualized with both sets to study the advantages of cropping the image data. With the music classification work, SVD was used to project the data onto few principal modes. Linear Discriminant Analysis (LDA) was used to train a classifier with training data, and the effectiveness of the classifier in classifying test data projected onto the same principal modes was calculated.

1 Introduction and Overview

SVD is a powerful method to transform higher-dimensional data in terms of its underlying major modes. While these principal modes are useful in representing the original higher-dimensional data in terms of fewer dimensions in the new, transformed basis, they are also helpful in studying the key features hidden in the data. In this work, we first use Yale faces dataset to illustrate the effectiveness of SVD in distinguishing important features from image data. Then, we build large datasets with audio data and use SVD in conjunction with discriminant analysis (LDA) to design a classifier capable of categorizing new music data.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

SVD transforms a matrix $\boldsymbol{A}$ into a series of matrix operations: first, a rotation by a unitary matrix $\boldsymbol{U}$, then a stretching action by a diagonal matrix $\boldsymbol{\Sigma}$, and then a rotation by a unitary matrix $\boldsymbol{V^*}$. For $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, the SVD is given by Eq. (1), where $\boldsymbol{U} \in \mathbb{C}^{m\times m}$ (row-space), $\boldsymbol{\Sigma} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{V} \in \mathbb{C}^{n\times n}$ [1].

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

The matrix operations could also be understood to be linear operations — the columns of $\boldsymbol{U}$, which represent the principal modes in the data $\boldsymbol{A}$, are first scaled by the diagonal element in the corresponding columns of $\boldsymbol{\Sigma}$; then the resulting columns are linearly combined with the elements in each column of $\boldsymbol{V^*}$ as coefficients (or weights), thus reproducing the original matrix $\boldsymbol{A}$ [2].

Since the singular values in $\boldsymbol{\Sigma}$ are arranged in a descending order of magnitude, it is easy to reduce a higher-dimensional system to a lower-dimensional representation by picking the first few modes (k) that comprise almost the entire energy contained in the original system. For the matrix $\boldsymbol{A}$ of rank r, Eq. (2) represents the matrix as a sum of its SVD modes, where $k \in [0, r]$, σ_j are the diagonal values of $\boldsymbol{\Sigma}$, $\boldsymbol{u}_j$ are columns of $\boldsymbol{U}$, and $\boldsymbol{v}_j^*$ are columns of $\boldsymbol{V}^*$.

$$\boldsymbol{A}_k = \sum_{j=1}^{k} \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^* \quad . \tag{2}$$

The most important property of SVD is that the lower-rank approximations constructed using Eq. (2) are also the *best* approximations, as determined by l^2 norm and Frobenius norm. These norms are given by Eq. (3) and Eq. (4).

*GitHub : `https://github.com/arvindhsharma`

$$\|\boldsymbol{A} - \boldsymbol{A}_k\|_2 = \sigma_{k+1} \quad . \tag{3}$$

$$\|\boldsymbol{A} - \boldsymbol{A}_k\|_F = \sqrt{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \cdots + \sigma_r^2} \quad . \tag{4}$$

2.2 Linear Discriminant Analysis (LDA)

LDA is a statistical technique used to classify different categories of data, based on a linear combination of features of the data. It relies on the availability of labels for the data along with the features, which is used to train the classifier, and this method falls under "supervised learning". LDA tries to find an optimal subspace where the given data can be well separated into different categories. In this projection, LDA tries to maximize the distance between the different categories of data while minimizing the space between points representing the same category.

When only two-classes are given, LDA works by finding a projection $\boldsymbol{w}$ as in Eq. (5), where the scatter matrix for inter-class data $\boldsymbol{S}_B$ is calculated using Eq. (6), and the scatter matrix for intra-class data $\boldsymbol{S}_W$ is calculated using Eq. (7), $\boldsymbol{x}$ are the vectors containing features for each class, and μ_1 and μ_2 are the means of the two classes.

$$\boldsymbol{w} = \arg\max_{\boldsymbol{w}} \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}} \quad . \tag{5}$$

$$\boldsymbol{S}_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \quad . \tag{6}$$

$$\boldsymbol{S}_W = \sum_{j=1}^{2} \sum_{\boldsymbol{x}} (\boldsymbol{x} - \mu_j)(\boldsymbol{x} - \mu_j)^T \quad . \tag{7}$$

The scatter matrices measure the variance of the data sets and variance of the difference between the means. Eq. (5), also known as *Rayleigh quotient*, can be calculated by solving the eigenvalue problem in Eq. (8), where λ is the largest eigenvalue and its associated eigenvector gives the projection basis. In this work, we use the pre-built LDA functionality in MATLAB to classify the data.

$$\boldsymbol{S}_B \boldsymbol{w} = \lambda \boldsymbol{S}_W \boldsymbol{w} \quad . \tag{8}$$

2.3 Wavelets

Wavelet decomposition is one type of multi-resolution decomposition for studying evolving frequency content in time-series data. Similar to Gabor transforms, they exploit the method of using filters (or windows) of various widths that slide across the time series data to get frequency content at different localized time instances.

3 Algorithm Implementation and Development

3.1 Yale Faces

Algorithm 1 provides an overview of the process to study the Yale Faces dataset. Here is a brief description of the process.

Algorithm 1 Algorithm to analyze Yale Faces data.

1: $Imgs\_cr \leftarrow$ Load images from cropped set
2: $Imgs\_uncr \leftarrow$ Load same number of random images from uncropped set as in $Imgs\_cr$
3: **procedure** SVD ANALYSIS(images)
4: $[U, S, V] \leftarrow$ SVD decomposition of images
5: Visualize modes of SVD by reshaping columns of U to the image size
6: Estimate the rank of SVD from the singular values in S
7: Calculate how much energy is contained in the SVD modes using norms
8: Visualize lower-rank approximations rebuilt from fewer modes in U, S, V
9: **end procedure**
10: Perform procedure (lines 3 — 9) on $Imgs\_cr$ and $Imgs\_uncr$

3.1.1 Data setup

The Yale faces database contains a vast number of images. For the purposes of this work, from the cropped image set, we load one standard image from 38 different subjects. The images have similar background light and facial expressions, are cropped to similar facial dimensions, and have same pixel dimensions. The images are transformed to column vectors, and are loaded as columns of a larger matrix, $Imgs\_cr$. Similarly, from the uncropped image dataset, we randomly pick the same number of images as were used to build $Imgs\_cr$, and build a matrix $Imgs\_uncr$, with the uncropped images as the columns. The uncropped images have subjects in various background lighting and facial expressions, and the subjects' faces are not necessarily centered in the middle of the images. We intentionally keep the number of columns of both matrices $Imgs\_cr$ and $Imgs\_uncr$ the same so that the comparisons between them are not skewed by large size differences.

3.1.2 SVD Analysis

SVD decomposition and analysis is performed on the two datasets. The left unitary matrix U contains the principal modes into which the data can be linearly split; visualizing the modes provides useful information about how well the modes can be used to reconstruct the images. The singular values in S provide information about the strength of the modes, and helps in estimating the true rank of the datasets. Here, we use the l^2 and Frobenius norms of representations reconstructed from fewer modes, normalized by the norms of the entire dataset, to analyze how many modes would be needed to build a good representation of the data. Finally, we visualize the lower-rank approximations to confirm the deductions from studies of singular values in S.

3.2 Music Classification

The process to build a classifier to identify music bands/genres is provided in Algorithm 2.

Algorithm 2 Algorithm to categorize different classes of music.

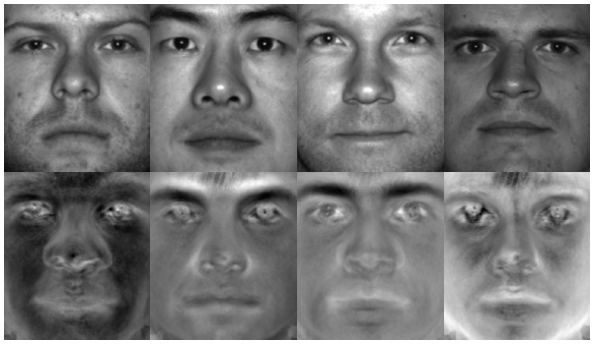
1: Collect 5 second music samples randomly from clips of each of the music categories
2: **procedure** WAVELET DECOMPOSITION(audio sample)
3: $sample\_mono \leftarrow$ Mean of audio channels in the MP3 sample
4: $resampled\_mono \leftarrow$ Resample $sample\_mono$ at 15 kHz
5: $wavelet \leftarrow$ Continuous wavelet transform of $resampled\_mono$
6: **end procedure**
7: $wavelets \leftarrow$ Perform procedure (lines 2 — 6) on each audio sample
8: $[Training\_set, Test\_set] \leftarrow$ Randomly assign 8/10 of wavelets to $Training\_set$ and the rest to $Test\_set$
9: $[U, S, V] \leftarrow$ SVD decomposition of $Training\_set$
10: $SVD\_proj = U^T * Training\_set$ ▷ Projection of training data onto SVD modes
11: Identify modes where different categories of music are separable in SVD-projection space
12: $Test\_data \leftarrow$ Projection of test set onto the identified principal modes
13: **for** n = 1: 1: 50 **do**
14: $[accuracy\_lin, accuracy\_quad] \leftarrow$ Accuracy of linear and quadratic classifiers, with 5/6 of training set as training data and rest as validation data
15: **end for**
16: Use all of training set projections to classify $test\_data$ and calculate accuracy

3.2.1 Data setup

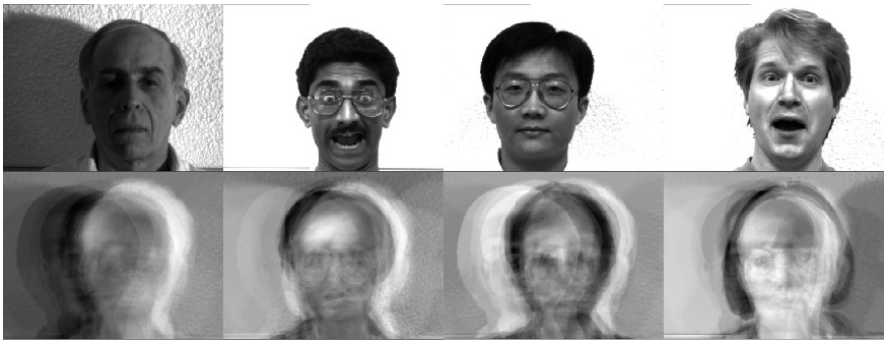
The first step is to collect music samples to build datasets. This is done by randomly sampling the available music files to collect 5 second samples. The same number of samples is collected for each of the music categories we are intending to classify. Instead of using the raw samples to build the classifier, we build wavelets from the samples since they provide a "snapshot" of the frequency and time content. SVD performs better on these "snapshot" like data instead of time-evolving signals like true audio data. About 8/10 of the wavelets from each category are used to build the training set, while the rest become the test set.

3.2.2 Projections onto SVD modes

The training set undergoes SVD to decompose the data into its principal modes. The left unitary matrix U contains the principal modes, and the projections of data onto these modes can be used to classify the music samples. First,



(a)



(b)

Figure 1: (a) Top row: montage showing four images from the cropped dataset; bottom row: first four modes of SVD reshaped to show the features in the dataset; (b) Top row: montage showing four images from the uncropped dataset; bottom row: first four modes of SVD.

we project the training set onto the modes in U, and visually identify the modes where the different categories are separable in space. Then, we project the test data onto these identified modes.

3.2.3 Classification

We randomly split 5/6 of the training data (projected onto the SVD modes of interest) into training set, and designate the rest as validation set. We run pre-built linear and quadratic classifiers in MATLAB to find the accuracy of each classifier, and repeat the process 50 times. This provides an estimate of the classifier's effectiveness. After that, we use all of the training data and the projections of test data onto the SVD modes to classify the samples in the test dataset.

4 Computational Results

4.1 Yale Faces

The top rows of Figure 1 (a) and 1(b) show the first four images used to build the cropped and uncropped datasets, respectively. The bottom rows of Figure 1 (a) and 1(b) show the first four dominant modes from the SVD of the respective datasets, from left to right. It can be readily observed that the cropped dataset has centered and well-aligned images, and produces modes that clearly identify major features in the faces in cropped dataset. For instance, the first mode shows a broad nose prominently, while the second mode shows a narrower nose and eyebrows more prominently. The uncropped dataset, on the other hand, has images of subjects in different expressions and positions. Therefore, it is unsurprising that the modes from SVD are rather hazy and do not show prominent identifiable features.

Figure 2 shows the results from analyses of the singular value matrix from the SVD. Figure 2(a) shows that while the cropped dataset has a prominent first mode and has a long tail of subsequent modes, the uncropped dataset has

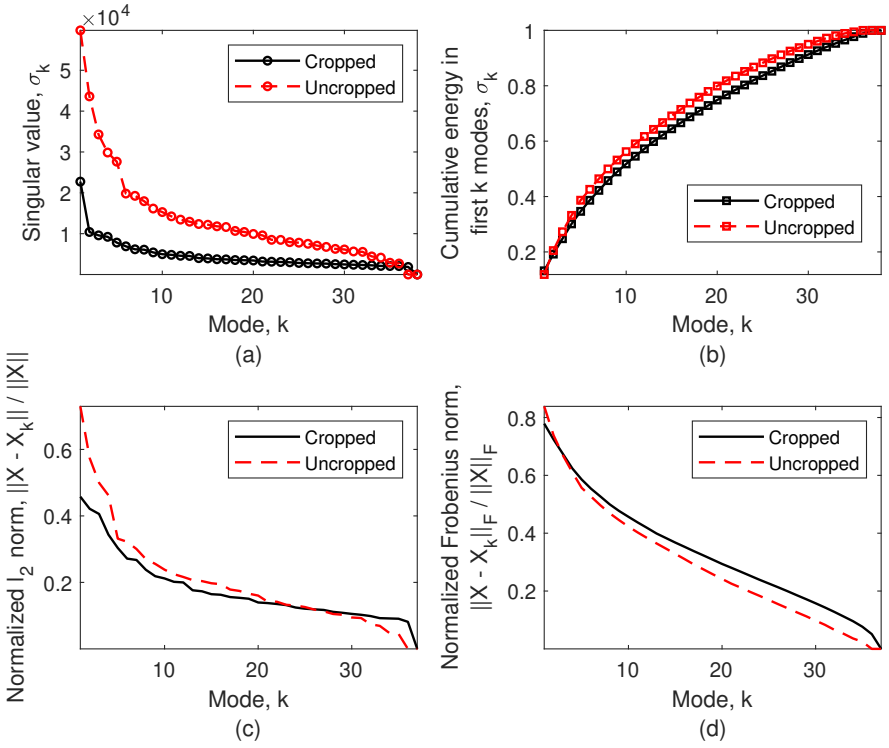


Figure 2: From SVD of cropped and uncropped datasets: (a) Singular values; (b) Cumulative sum of singular values normalized by the total sum of singular values; (c) l^2 norm of low-rank approximations normalized by l^2 of corresponding matrices; (d) Frobenius norm of low-rank approximations normalized by Frobenius norm of corresponding matrices.

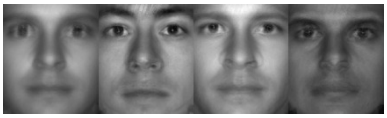
about five major modes, with the rest of the modes forming a longer tail. Figure 2(b) shows the cumulative strength of the first few modes of each dataset, normalized by the total strength. It is hard to distinguish between the two datasets using this metric, however. Figure 2(c) is a plot of the l^2 norm of lower-rank approximations normalized by the l^2 norm of the entire dataset. Since this metric is similar to the error from the true representation, it is clear from the plot that the cropped dataset is more representative of the actual data with fewer modes than what the uncropped dataset requires to produce a similar level of representation. Figure 2(d) is similar to Figure 2(c), except that the Frobenius norm is used in place of l^2. This plot shows that going by the Frobenius norm, the cropped dataset can be better represented with only first two modes, compared to the first two-mode representation of the uncropped dataset. However, the plot shows that the uncropped dataset has a better representation with fewer modes than cropped dataset as number of modes increases beyond two. This discrepancy in the analysis can be resolved by looking at lower-order approximations, and will be discussed later.

The rows of the right unitary matrix, $\boldsymbol{V}*$, are proportional to the coefficients of the linear combination that would be required to re-form the images in the datasets using the columns of the modes in the left unitary matrix, $\boldsymbol{U}$. Figure 8 (a), 8(c), and 8(e) (in Appendix C) show how the first three rows of $\boldsymbol{V}*$ from the cropped dataset vary, corresponding to the 38 images in the dataset. Figure 8 (b), 8(d), and 8(f) show the corresponding rows from $\boldsymbol{V}*$ of the uncropped dataset.

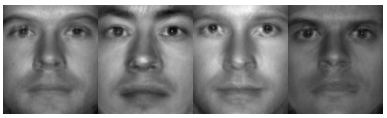
Figure 3 and 4 show the lower-rank approximations of the cropped and uncropped datasets respectively. It is seen that with just 24 to 28 modes from the SVD, the cropped dataset images are well-represented. However, even with 32 modes, the uncropped dataset is not well represented by the reconstructions, as can be seen in the poor reconstruction of spectacles in the second image in Figure 4(f). The norms in Figure 2(c) and 2(d) did not adequately predict this, and this could be due to the different image sizes between the two datasets.

4.2 Music Classification

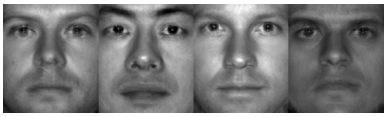
The music classification procedure in Algorithm 2 is used to classify three different cases: Case 1 $\rightarrow$ 3 bands from 3 different genres; Case 2 $\rightarrow$ 3 bands from the same genre; and Case 3 $\rightarrow$ 3 genres encompassing multiple bands. In each case, the samples are first transformed to wavelets (using a Morse wavelet) and the training set is then are decomposed using SVD. Figure 5(a)—5(d) show spectrograms of the first four modes of the decomposition for Case 1, which are



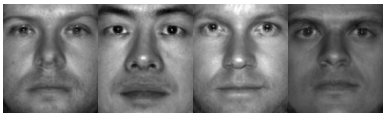
(a) Rank = 5



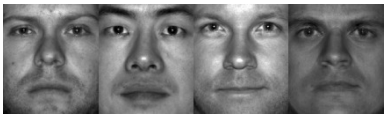
(b) Rank = 12



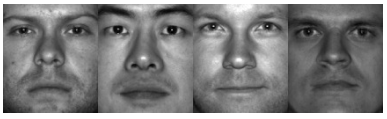
(c) Rank = 20



(d) Rank = 24

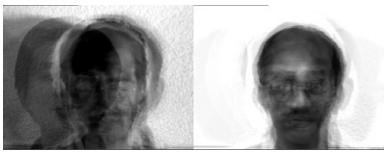


(e) Rank = 28

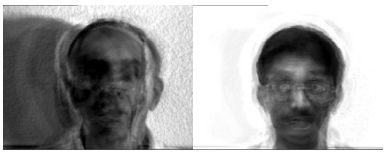


(f) Rank = 32

Figure 3: For the cropped dataset: (a)—(f) show reconstructions of four images (same as in top row of Figure 1(a)) with ranks 5, 12, 20, 24, 28, and 32, respectively.



(a) Rank = 5



(b) Rank = 12

(c) Rank = 20

(d) Rank = 24

(e) Rank = 28

(f) Rank = 32

Figure 4: For the uncropped dataset: (a)—(f) show reconstructions of two images (same as in top row of Figure 1(b)) with ranks 5, 12, 20, 24, 28, and 32, respectively.

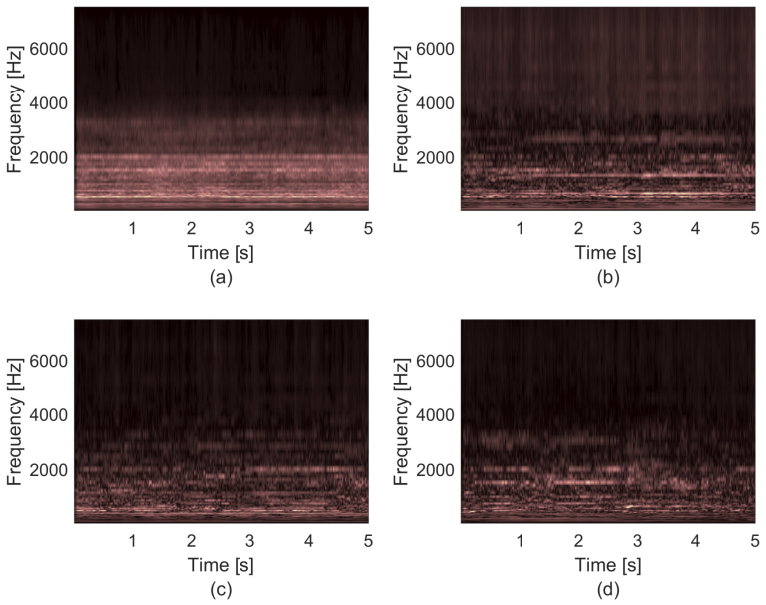


Figure 5: For Case 1 (3 bands, 3 different genres): (a)—(d) show the first four modes of the SVD of wavelet transformed training set. These are the principal features in the wavelet transform of the music signals.

extracted from the first four columns of the matrix U from SVD. The data is then projected onto these modes, and the resulting projected data in each mode is plotted as histograms in Figure 6(a)—6(f). This helps in identifying the modes where the domains of the histograms are separable. For Case 1, modes 1, 2, and 6 offer good separation, and this is verified by forming a 3 dimensional space using these 3 modes as axes and plotting the projections, as in Figure 9 (in Appendix C). For Case 2 and Case 3, we use the first 5 modes to project the data since this was experimentally found to be more robust.

Linear and quadratic classifiers are built using MATLAB's `classify()` function, with the SVD projections as the training data and suitable labels for the three categories. The training data is split into two sets to form training and validation data, and the the classifiers' accuracies are recorded. As an illustration, Figure 7 shows the accuracies over fifty trials for Case 1. In the subsequent step, the classifiers are used with all the training data and their accuracy is recorded with the test data. The results are provided in Table 1.

Test Case	Classifier Type	Accuracy [%]			
		Training/Validation Data			Test Data
		Mean	Minimum	Maximum	
3 bands from 3 different genres	Linear	81	58	100	94
	Quadratic	82	71	96	94
3 bands from same genre	Linear	65	46	79	36
	Quadratic	68	58	92	61
3 different genres (multiple bands)	Linear	77	67	88	81
	Quadratic	76	67	88	81

Table 1: Computational results - music classification.

It can be seen from the results that the classifiers are more successful in accurately identifying 3 bands from 3 different genres. The accuracy falls when the classifiers try to distinguish between 3 bands from the same genre, as can be expected. The accuracy improves significantly when the classifier tries to categorize music broadly into 3 different genres. Moreover, the quadratic classifier is observed to be more successful generally than the linear classifier.

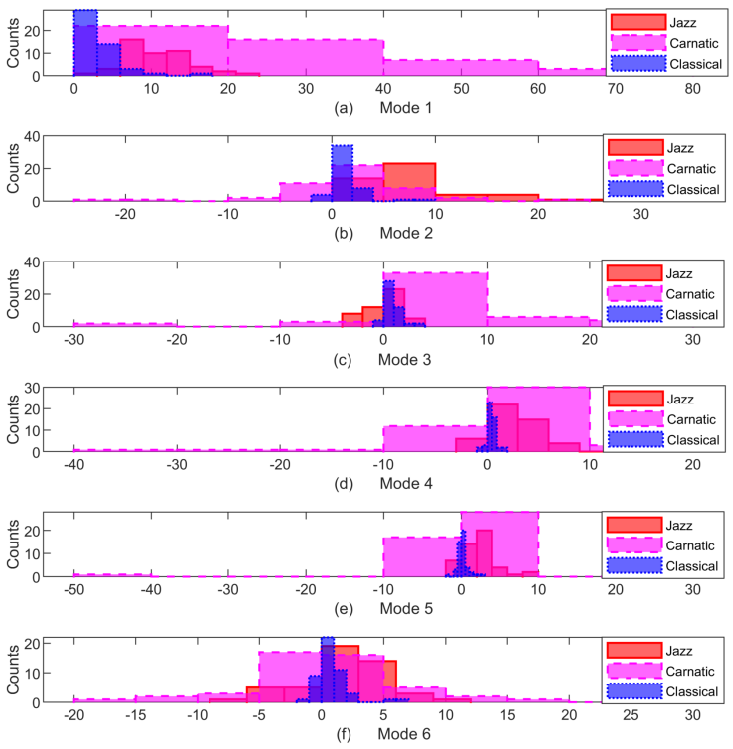


Figure 6: For Case 1 (3 bands, 3 different genres): (a)—(f) show the histograms of training data projected onto the first six modes.

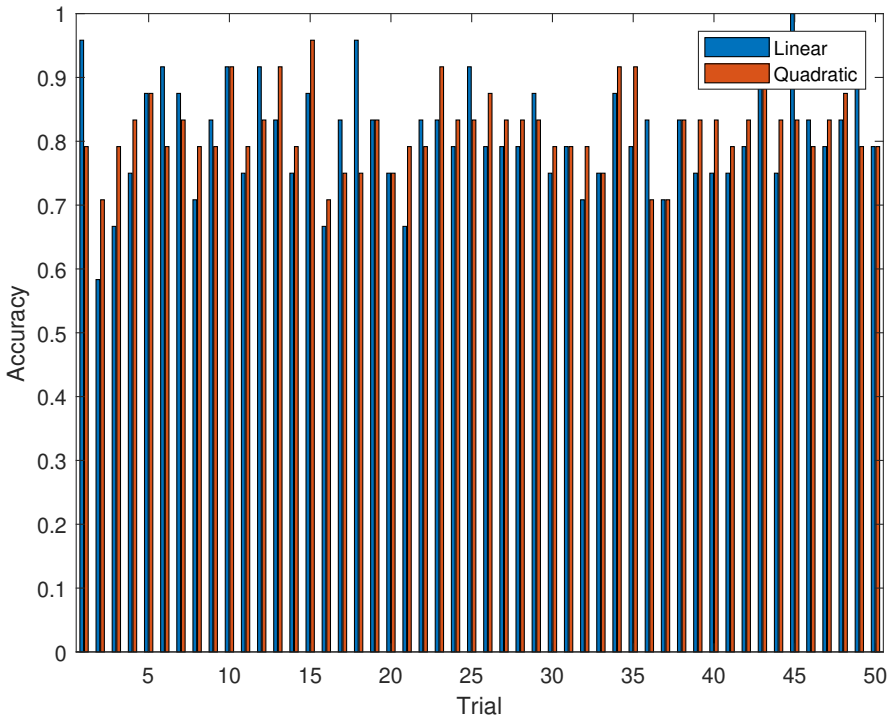


Figure 7: For Case 1 (3 bands, 3 different genres): accuracy of the linear and quadratic classifiers with random train/validation data splits for 50 trials.

5 Summary and Conclusions

This work illustrated the use of SVD in identifying key features in higher-dimensional datasets, and how this could be used to characterize the data. In the case of Yale images, it was seen that good pre-processing techniques to align the data is critical in successful implementation of SVD, since the SVD produced well-defined features with cropped data but failed to do so with uncropped data. Moreover, a good representation of the higher-dimensional data is possible with fewer modes. In the case of music classification, projections of data on to SVD modes were used to distinguish between different datasets. Along with LDA, this is an effective technique to categorize new data after building a classifier using representative datasets. This work also showed the challenges in distinguishing between the datasets when the categories are similar to each other.

References

[1] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

[2] Robert A. Beezer. *A First Course in Linear Algebra*. URL: http://linear.ups.edu/html/fcla.html. (accessed: 03/08/2020).

Appendix A MATLAB Functions

Here is an overview of the major functions used in the programs in Appendix B.

- `[U, S, V] = svd(A)` returns the singular value decomposition of the matrix `A`, such that `A = U * S * V'`.
- `B = reshape(A, sz)` reshapes the matrix `A` to the size specified by the size vector `sz`, and outputs the result in `B`.
- `y = resample(x, p, q)` resamples the data in column vector `x` (or columns of `x` if it is a matrix) at a new rate `(p / q)` times the original sampling rate. It also applies an anti-aliasing FIR lowpass filter and compensates for the time delay introduced by the filtering process.
- `[wt, f] = cwt(x, options` returns the continuous wavelet transform of `x` in the variable `wt`, and the corresponding frequencies in `f`. Options such as the wavelet name and construction can be passed to the command.
- `class = classify(sample, training, group, 'type'` builds a discriminant from the data in `training` and associated labels in `group`, and categorizes the data in `sample` into the trained groups. The type of discriminant (`linear, quadratic, etc.`) could be passed in the `'type'` option.

Appendix B MATLAB Code

B.1 SVD Analysis of Yale Faces

```
%% YALE FACES

%% LOAD CROPPED DATA
clear; clc; close all

addpath('D:\course_work\amath582\hw4\yalefaces_cropped\CroppedYale')
yale_cr_folders = dir(['D:\course_work\amath582\hw4\',...
    'yalefaces_cropped\CroppedYale\yale*']);

Imgs_cr = nan(192 * 168, length(yale_cr_folders));
Imgs_cr_montage = nan(192, 168, length(yale_cr_folders));

% Get one image from the 38 different subjects
for ii = 1:1:length(yale_cr_folders)
    filename = strcat(yale_cr_folders(ii).folder, '\', ...
        yale_cr_folders(ii).name, '\', yale_cr_folders(ii).name, ...
```

```
        '_P00A+000E+00.pgm');
    temp = imread(filename);
    Imgs_cr_montage(:, :, ii) = temp;
    Imgs_cr(:, ii) = temp(:);
end

figure, montage(uint8(Imgs_cr_montage))

mean_face_cr = mean(Imgs_cr, 2);
X_Imgs_cr = Imgs_cr - mean_face_cr;

%% SVD
[U_cr, S_cr, V_cr] = svd(X_Imgs_cr, 'econ');
sig_cr = diag(S_cr);

U_cr_rs = nan(size(U_cr));
for jj = 1:1:length(yale_cr_folders)
    U_cr_rs(:, jj) = rescale(U_cr(:, jj), 0, 255);
end
U_cr_rs_mon = reshape(U_cr_rs, [192, 168, length(yale_cr_folders)]);
U_cr_rs_mon = cat(3, U_cr_rs_mon);
figure, montage(uint8(U_cr_rs_mon))

Y_cr = V_cr * X_Imgs_cr';

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];

s1 = subplot(2, 2, 1);
s1.Box = 'on';
hold on
h1 = plot(sig_cr, 'ko-', 'LineWidth', 1.1);
h1.MarkerSize = 3.5;
xlabel({'Mode, k'; '(a)'})
ylabel('Singular value, \sigma_k')
axis tight

s2 = subplot(2, 2, 2);
s2.Box = 'on';
hold on
h2 = plot(cumsum(sig_cr) / sum(sig_cr), 'ks-', 'LineWidth', 1.1);
h2.MarkerSize = 3.5;
xlabel({'Mode, k'; '(b)'})
ylabel({'Cumulative energy in', 'first k modes, \sigma_k'})
axis tight

%% LOAD UNCROPPED DATA

addpath('D:\course_work\amath582\hw4\yalefaces_uncropped\yalefaces\')
yale_uncr_folders = dir(['D:\course_work\amath582\hw4\',...
    'yalefaces_uncropped\yalefaces\sub*']);

% Randomly select 38 images (same as cropped dataset) from the dataset
Imgs_uncr = nan(243 * 320, length(yale_cr_folders));
```

```
Imgs_uncr_montage = nan(243, 320, length(yale_cr_folders));
rand_files = randperm(length(yale_uncr_folders), length(yale_cr_folders));

c1 = 0;
for ii = rand_files
    c1 = c1 + 1;
    filename = strcat(yale_uncr_folders(ii).folder, '\', ...
        yale_uncr_folders(ii).name);
    temp = imread(filename);
    Imgs_uncr_montage(:, :, c1) = temp;
    Imgs_uncr(:, c1) = temp(:);
end

figure, montage(uint8(Imgs_uncr_montage))

mean_face_uncr = mean(Imgs_uncr, 2);
X_Imgs_uncr = Imgs_uncr - mean_face_uncr;

%% SVD
[U_uncr, S_uncr, V_uncr] = svd(X_Imgs_uncr, 'econ');
sig_uncr = diag(S_uncr);


U_uncr_rs = nan(size(U_uncr));
for jj = 1:1:length(yale_cr_folders)
    U_uncr_rs(:, jj) = rescale(U_uncr(:, jj), 0, 255);
end
U_uncr_rs_mon = reshape(U_uncr_rs, [243, 320, length(yale_cr_folders)]);
U_uncr_rs_mon = cat(3, U_uncr_rs_mon);
figure, montage(uint8(U_uncr_rs_mon))

Y_uncr = V_uncr * X_Imgs_uncr';

figure(fig);
subplot(s1);
h3 = plot(sig_uncr, 'ro--', 'LineWidth', 1.1);
h3.MarkerSize = 3.5;
axis tight
legend([h1, h3], {'Cropped', 'Uncropped'})

subplot(s2);
h4 = plot(cumsum(sig_uncr) / sum(sig_uncr), 'rs--', 'LineWidth', 1.1);
h4.MarkerSize = 3.5;
axis tight
legend([h2, h4], {'Cropped', 'Uncropped'})

%% VISUALIZING THE IMAGES AND SVD MODES

% Mode
cr_imgs = cat(3, uint8(Imgs_cr_montage(:, :, 1:4)),...
    uint8(U_cr_rs_mon(:, :, 1:4)));
uncr_imgs = cat(3, uint8(Imgs_uncr_montage(:, :, 1:4)),...
    uint8(U_uncr_rs_mon(:, :, 1:4)));
fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
```

```
fig.PaperSize = [6.75 5.0625];
sa = subplot(2, 1, 1);
montage(cr_imgs, 'Size', [2, 4])
xlabel('(a)')
sb = subplot(2, 1, 2);
montage(uncr_imgs, 'Size', [2, 4])
xlabel('(b)')

% Mode strengths
fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
plot_labels = {'(a)', '(b)', '(c)', '(d)', '(e)', '(f)', '(g)', '(h)'};
for mm = 1: 1: 3
    sc = subplot(3, 2, (mm - 1) * 2 + 1);
    h = plot(V_cr(:, mm), 'k.-', 'LineWidth', 1.1);
    xlabel(['Mode ', num2str(mm), '      ', plot_labels{(mm - 1) * 2 + 1}])
    ylabel('Magnitude')
    h.MarkerSize = 10;
    sd = subplot(3, 2, (mm - 1) * 2 + 2);
    h = plot(V_uncr(:, mm), 'k.-', 'LineWidth', 1.1);
    xlabel(['Mode ', num2str(mm), '      ', plot_labels{(mm - 1) * 2 + 2}])
    ylabel('Magnitude')
    h.MarkerSize = 10;
end

%% IMAGE RECONSTRUCTION

fig = gobjects([2, 1]);
fig(1) = figure;
fig(2) = figure;
fig(1).Units = 'inches';
fig(1).Position = [-.1 1.8 6.75 6];
fig(1).PaperUnits = 'inches';
fig(1).PaperSize = [6.75 6];
fig(2).Units = 'inches';
fig(2).Position = [-.1 1.8 6.75 6];
fig(2).PaperUnits = 'inches';
fig(2).PaperSize = [6.75 6];
r = [5, 12, 20, 24, 28, 32];
I_recon_cr = nan(192, 168, 32);
I_recon_uncr = nan(243, 320, 32);
for ii = 1: 1: length(r)
    temp_cr = mean_face_cr +...
        U_cr(:, 1:r(ii)) * S_cr(1:r(ii), 1:r(ii)) * V_cr(:, 1:r(ii))';
    temp_uncr = mean_face_uncr +...
        U_uncr(:, 1:r(ii)) * S_uncr(1:r(ii), 1:r(ii)) * V_uncr(:, 1:r(ii))';
    for jj = 1: 1: 4
        I_recon_cr(:, :, (ii - 1) * 4 + jj) = reshape(temp_cr(:, jj), ...
            [192, 168]);
        I_recon_uncr(:, :, (ii - 1) * 4 + jj) = reshape(temp_uncr(:, jj), ...
            [243, 320]);
        figure(fig(1));
        s5 = subplot(3, 2, ii);
        montage(uint8(I_recon_cr(:, :, (ii - 1) * 4 + (1:4))), 'Size', [1, 4])
```

```
        xlabel([plot_labels{ii}, '      ', 'Rank = ', num2str(r(ii))])
        figure(fig(2));
        s6 = subplot(3, 2, ii);
        montage(uint8(I_recon_uncr(:, :, (ii - 1) * 4 + (1:2))), 'Size', [1, 2])
        xlabel([plot_labels{ii}, '      ', 'Rank = ', num2str(r(ii))])
    end
end

%% COMPARING L2 AND FROBENIUS NORMS

l2_cr = nan(37, 1);
l2_uncr = nan(37, 1);
fro_cr = nan(37, 1);
fro_uncr = nan(37, 1);

for ii = 1:1:37
fro_cr(ii) = norm(sig_cr(ii + 1:end));
fro_uncr(ii) = norm(sig_uncr(ii + 1:end));
end
l2_cr_norm = sig_cr(2:end) / sig_cr(1);
l2_uncr_norm = sig_uncr(2:end) / sig_uncr(1);
fro_cr_norm = fro_cr / norm(sig_cr);
fro_uncr_norm = fro_uncr / norm(sig_uncr);

figure(fig)
s3 = subplot(2, 2, 3);
s3.Box = 'on';
h5 = plot(l2_cr_norm, 'k', 'LineWidth', 1.1);
hold on
h6 = plot(l2_uncr_norm, 'r--', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(c)'})
ylabel('Normalized l_2 norm, ||X - X_k|| / ||X||')
axis tight
legend([h5, h6], {'Cropped', 'Uncropped'})


s4 = subplot(2, 2, 4);
s4.Box = 'on';
hold on
h7 = plot(fro_cr_norm, 'k', 'LineWidth', 1.1);
h8 = plot(fro_uncr_norm, 'r--', 'LineWidth', 1.1);
xlabel({'Mode, k'; '(d)'})
ylabel({'Normalized Frobenius norm,', '||X - X_k||_F / ||X||_F'})
axis tight
legend([h7, h8], {'Cropped', 'Uncropped'})
```

B.2 Music Classification — Case 1: 3 Bands, 3 Genres

```
%% MUSIC CLASSIFICATION - 3 BANDS, 3 GENRES

clear; clc; close all

%% BUILDING TRAINING AND TESTING DATASETS

addpath(genpath('D:\course_work\amath582\hw4\music_data\'))

jazz_files = dir(['D:\course_work\amath582\hw4\music_data\',...
```

```
    'jazz\keithmitchell_*']);
carnatic_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'carnatic\semma*']);
classical_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'classical\Beethoven*']);

all_files = cell(3, 1);
[all_files{1}, all_files{2}, all_files{3}] = deal(...
    extractfield(jazz_files, 'name'), extractfield(carnatic_files,...
    'name'), extractfield(classical_files, 'name'));

Fs_red = 15e3;  % Reduced sampling rate
% size = (wvlt_decomp * num_of_samples * num_of_files/case, num_of_cases)
training_set = nan(75e5, 144);
test_set = nan(75e5, 36);
for ii = 1: 1: 3
    files = all_files{ii};
    files_order = randperm(length(files));  % Randomly pick files to train on
    for jj = 1: 1: 10
        filename = files{files_order(jj)};
        if jj <= 8
            training_set(:, (ii - 1) * 48 + (jj - 1) * 6 + (1:6)) =...
                wave_decomp(filename, 6, Fs_red);
        else
            test_set(:, (ii - 1) * 12 + (jj - 9) * 6 + (1:6)) =...
                wave_decomp(filename, 6, Fs_red);
        end
    end
end

%% SVD OF TRAINING DATA

mean_training_set = mean(training_set, 2);
[U, S, V] = svd(training_set - mean_training_set, 'econ');

% figure,
% plot(diag(S), 'ko')

% SVD Modes

[~, freq] = wave_decomp(filename, 1, Fs_red);
fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
plot_labels = {'(a)', '(b)', '(c)', '(d)'};
for mm = 1: 1: 4
    s = subplot(2, 2, mm);
    pcolor((1: Fs_red * 5) / Fs_red, freq, reshape(abs(U(:, mm)),...
        [100, Fs_red * 5]))
    shading interp
    axis tight
    colormap(pink)
    xlabel({'Time [s]'; plot_labels{mm}})
    ylabel('Frequency [Hz]')
end
```

```
% Projection of data onto principal modes

SVD_proj = (U' * training_set)';

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];
plot_labels = {'(a)', '(b)', '(c)', '(d)', '(e)', '(f)'};
for mm = 1:1:6
    s1 = subplot(6, 1, mm);
    s1.Box = 'on';
    hold on
    h1 = histogram((SVD_proj(1:48, mm)), 'EdgeColor', 'r', 'FaceColor', 'r');
    h2 = histogram((SVD_proj(49:96, mm)), 'EdgeColor', 'm', 'FaceColor', 'm');
    h3 = histogram((SVD_proj(97:144, mm)), 'EdgeColor', 'b', 'FaceColor', 'b');
    h2.LineStyle = '--'; h3.LineStyle = ':';
    h1.LineWidth = 1.1; h2.LineWidth = 1.1; h3.LineWidth = 1.1;
    xlabel(plot_labels{mm})
    ylabel('Counts')
    legend('Jazz', 'Carnatic', 'Classical')
end

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
ax = gca;
ax.Box = 'on';
ax.View = [11.412, 63.0438];
hold on
h1 = plot3(SVD_proj(1:48, 1), SVD_proj(1:48, 2), SVD_proj(1:48, 6), 'ro');
h2 = plot3(SVD_proj(49:96, 1), SVD_proj(49:96, 2), SVD_proj(49:96, 6), 'm^');
h3 = plot3(SVD_proj(97:144, 1), SVD_proj(97:144, 2), SVD_proj(97:144, 6), 'bp');
xlabel('PCA 1'), ylabel('PCA 2'), zlabel('PCA 6')
legend('Jazz', 'Carnatic', 'Classical')

% figure;
% for mm = 1:1:6
% subplot(6, 3, (mm - 1) * 3 + 1)
% plot(1:48, (SVD_proj(1:48, mm)), 'ko-')
% subplot(6, 3, (mm - 1) * 3 + 2)
% plot(1:48, (SVD_proj(49:96, mm)), 'ko-')
% subplot(6, 3, (mm - 1) * 3 + 3)
% plot(1:48, (SVD_proj(97:144, mm)), 'ko-')
% end

%% LDA WITH CROSS VALIDATION

labels = [-1 * ones(40, 1);...
          0 * ones(40, 1);...
          1 * ones(40, 1)];
truth = [-1 * ones(8, 1);...
          0 * ones(8, 1);...
```

```
          1 * ones(8, 1)];
accuracy_lin = nan(length(truth), 1);
accuracy_quad = nan(length(truth), 1);
for nn = 1: 1: 50
    rand_vec = randperm(48);
    train_mat = [SVD_proj(rand_vec(1:40), [1, 2, 6]);...
                 SVD_proj(rand_vec(1:40) + 48, [1, 2, 6]);...
                 SVD_proj(rand_vec(1:40) + 96, [1, 2, 6])];
    test_mat = [SVD_proj(rand_vec(41:48), [1, 2, 6]);...
                SVD_proj(rand_vec(41:48) + 48, [1, 2, 6]);...
                SVD_proj(rand_vec(41:48) + 96, [1, 2, 6])];
    class_lin = classify(test_mat, train_mat, labels);
    class_quad = classify(test_mat, train_mat, labels, 'quadratic');
    accuracy_lin(nn, 1) = sum(class_lin == truth) / length(truth);
    accuracy_quad(nn, 1) = sum(class_quad == truth) / length(truth);
end

fprintf('\nTraining data:\n')
fprintf('Quadratic discriminant accuracy\n')
fprintf('Mean = %.2f \t Minimum = %.2f \t Maximum = %.2f\n',...
    mean(accuracy_quad), min(accuracy_quad), max(accuracy_quad))
fprintf('Linear discriminant accuracy\n')
fprintf('Mean = %.2f \t Minimum = %.2f \t Maximum = %.2f\n',...
    mean(accuracy_lin), min(accuracy_lin), max(accuracy_lin))

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
ax = gca;
ax.Box = 'on';
h1 = bar([accuracy_lin, accuracy_quad]);
legend('Linear', 'Quadratic')
xlabel('Trial')
ylabel('Accuracy')

%% LDA ON TEST DATA

test_truth = [-1 * ones(12, 1);...
              0 * ones(12, 1);...
              1 * ones(12, 1)];
test_mat =(U(:, [1, 2, 6])' * test_set)';
labels = [-1 * ones(48, 1);...
          0 * ones(48, 1);...
          1 * ones(48, 1)];
train_mat = [SVD_proj(1:48, [1, 2, 6]);
             SVD_proj(49:96, [1, 2, 6]);
             SVD_proj(97:144, [1, 2, 6])];
class_lin = classify(test_mat, train_mat, labels);
class_quad = classify(test_mat, train_mat, labels, 'quadratic');
accuracy_lin = sum(class_lin == test_truth) / length(test_truth);
accuracy_quad = sum(class_quad == test_truth) / length(test_truth);

fprintf('\nTest data:\n')
fprintf('Quadratic discriminant\n')
fprintf('Accuracy = %.2f\n', accuracy_quad)
```

```
fprintf('Linear discriminant\n')
fprintf('Accuracy = %.2f\n', accuracy_lin)
```

B.3 Music Classification — Case 2: 3 Bands, Same Genre

```
%% MUSIC CLASSIFICATION - 3 BANDS, 1 GENRE

clear; clc; close all

%% BUILDING TRAINING AND TESTING DATASETS

addpath(genpath('D:\course_work\amath582\hw4\music_data\'))

carnatic1_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'carnatic\dkj_*']);
carnatic2_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'carnatic\semma*']);
carnatic3_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'carnatic\rk*']);

all_files = cell(3, 1);
[all_files{1}, all_files{2}, all_files{3}] = deal(...
    extractfield(carnatic1_files, 'name'), extractfield(carnatic2_files,...
    'name'), extractfield(carnatic3_files, 'name'));

Fs_red = 15e3;  % Reduced sampling rate
% size = (wvlt_decomp * num_of_samples * num_of_files/case, num_of_cases)
training_set = nan(75e5, 144);
test_set = nan(75e5, 36);
for ii = 1: 1: 3
    files = all_files{ii};
    files_order = randperm(length(files));  % Randomly pick files to train on
    for jj = 1: 1: 10
        filename = files{files_order(jj)};
        if jj <= 8
            training_set(:, (ii - 1) * 48 + (jj - 1) * 6 + (1:6)) =...
                wave_decomp(filename, 6, Fs_red);
        else
            test_set(:, (ii - 1) * 12 + (jj - 9) * 6 + (1:6)) =...
                wave_decomp(filename, 6, Fs_red);
        end
    end
end

%% SVD OF TRAINING DATA

mean_training_set = mean(training_set, 2);
[U, S, V] = svd(training_set - mean_training_set, 'econ');

% figure,
% plot(diag(S), 'ko')

% SVD Modes

[~, freq] = wave_decomp(filename, 1, Fs_red);
fig = figure;
fig.Units = 'inches';
```

```
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
plot_labels = {'(a)', '(b)', '(c)', '(d)'};
for mm = 1: 1: 4
    s = subplot(2, 2, mm);
    pcolor((1: Fs_red * 5) / Fs_red, freq, reshape(abs(U(:, mm)),...
        [100, Fs_red * 5]))
    shading interp
    axis tight
    colormap(pink)
    xlabel({'Time [s]'; plot_labels{mm}})
    ylabel('Frequency [Hz]')
end

% Projection of data onto principal modes

SVD_proj = (U' * training_set)';

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];
plot_labels = {'(a)', '(b)', '(c)', '(d)', '(e)', '(f)'};
for mm = 1:1:6
    s1 = subplot(6, 1, mm);
    s1.Box = 'on';
    hold on
    h1 = histogram((SVD_proj(1:48, mm)), 'EdgeColor', 'r', 'FaceColor', 'r');
    h2 = histogram((SVD_proj(49:96, mm)), 'EdgeColor', 'm', 'FaceColor', 'm');
    h3 = histogram((SVD_proj(97:144, mm)), 'EdgeColor', 'b', 'FaceColor', 'b');
    h2.LineStyle = '--'; h3.LineStyle = ':';
    h1.LineWidth = 1.1; h2.LineWidth = 1.1; h3.LineWidth = 1.1;
    xlabel(plot_labels{mm})
    ylabel('Counts')
    legend('Jazz', 'Carnatic', 'Classical')
end

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
ax = gca;
ax.Box = 'on';
ax.View = [11.412, 63.0438];
hold on
h1 = plot3(SVD_proj(1:48, 2), SVD_proj(1:48, 4), SVD_proj(1:48, 6), 'ro');
h2 = plot3(SVD_proj(49:96, 2), SVD_proj(49:96, 4), SVD_proj(49:96, 6), 'm^');
h3 = plot3(SVD_proj(97:144, 2), SVD_proj(97:144, 4), SVD_proj(97:144, 6), 'bp');
xlabel('PCA 1'), ylabel('PCA 2'), zlabel('PCA 6')
legend('Jazz', 'Carnatic', 'Classical')

% figure;
% for mm = 1:1:6
% subplot(6, 3, (mm - 1) * 3 + 1)
```

```
% plot(1:48, (SVD_proj(1:48, mm)), 'ko-')
% subplot(6, 3, (mm - 1) * 3 + 2)
% plot(1:48, (SVD_proj(49:96, mm)), 'ko-')
% subplot(6, 3, (mm - 1) * 3 + 3)
% plot(1:48, (SVD_proj(97:144, mm)), 'ko-')
% end

%% LDA WITH CROSS VALIDATION

labels = [-1 * ones(40, 1);...
          0 * ones(40, 1);...
          1 * ones(40, 1)];
truth = [-1 * ones(8, 1);...
          0 * ones(8, 1);...
          1 * ones(8, 1)];
accuracy_lin = nan(length(truth), 1);
accuracy_quad = nan(length(truth), 1);
for nn = 1: 1: 50
    rand_vec = randperm(48);
    train_mat = [SVD_proj(rand_vec(1:40), 1:5);...
                 SVD_proj(rand_vec(1:40) + 48, 1:5);...
                 SVD_proj(rand_vec(1:40) + 96, 1:5)];
    test_mat = [SVD_proj(rand_vec(41:48), 1:5);...
                SVD_proj(rand_vec(41:48) + 48, 1:5);...
                SVD_proj(rand_vec(41:48) + 96, 1:5)];
    class_lin = classify(test_mat, train_mat, labels);
    class_quad = classify(test_mat, train_mat, labels, 'quadratic');
    accuracy_lin(nn, 1) = sum(class_lin == truth) / length(truth);
    accuracy_quad(nn, 1) = sum(class_quad == truth) / length(truth);
end

fprintf('\nTraining data:\n')
fprintf('Quadratic discriminant accuracy\n')
fprintf('Mean = %.2f \t Minimum = %.2f \t Maximum = %.2f\n',...
    mean(accuracy_quad), min(accuracy_quad), max(accuracy_quad))
fprintf('Linear discriminant accuracy\n')
fprintf('Mean = %.2f \t Minimum = %.2f \t Maximum = %.2f\n',...
    mean(accuracy_lin), min(accuracy_lin), max(accuracy_lin))

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
ax = gca;
ax.Box = 'on';
h1 = bar([accuracy_lin, accuracy_quad]);
legend('Linear', 'Quadratic')
xlabel('Trial')
ylabel('Accuracy')

%% LDA ON TEST DATA

test_truth = [-1 * ones(12, 1);...
              0 * ones(12, 1);...
              1 * ones(12, 1)];
test_mat =(U(:, 1:5)' * test_set)';
```

```
labels = [-1 * ones(48, 1);...
          0 * ones(48, 1);...
          1 * ones(48, 1)];
train_mat = [SVD_proj(1:48, 1:5);
             SVD_proj(49:96, 1:5);
             SVD_proj(97:144, 1:5)];
class_lin = classify(test_mat, train_mat, labels);
class_quad = classify(test_mat, train_mat, labels, 'quadratic');
accuracy_lin = sum(class_lin == test_truth) / length(test_truth);
accuracy_quad = sum(class_quad == test_truth) / length(test_truth);

fprintf('\nTest data:\n')
fprintf('Quadratic discriminant\n')
fprintf('Accuracy = %.2f\n', accuracy_quad)
fprintf('Linear discriminant\n')
fprintf('Accuracy = %.2f\n', accuracy_lin)
```

B.4 Music Classification — Case 3: 3 Genres

```
%% MUSIC CLASSIFICATION - 3 GENRES

clear; clc; close all

%% BUILDING TRAINING AND TESTING DATASETS

addpath(genpath('D:\course_work\amath582\hw4\music_data\'))

jazz_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'jazz\']);
carnatic_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'carnatic\']);
classical_files = dir(['D:\course_work\amath582\hw4\music_data\',...
    'classical\']);

all_files = cell(3, 1);
[all_files{1}, all_files{2}, all_files{3}] = deal(...
    extractfield(jazz_files(3:end), 'name'),...
    extractfield(carnatic_files(3:end), 'name'),...
    extractfield(classical_files(3:end), 'name'));

Fs_red = 15e3;  % Reduced sampling rate
% size = (wvlt_decomp * num_of_samples * num_of_files/case, num_of_cases)
training_set = nan(75e5, 144);
test_set = nan(75e5, 36);
for ii = 1: 1: 3
    files = all_files{ii};
    files_order = randperm(length(files));  % Randomly pick files to train on
    for jj = 1: 1: 10
        filename = files{files_order(jj)};
        if jj <= 8
            training_set(:, (ii - 1) * 48 + (jj - 1) * 6 + (1:6)) =...
                wave_decomp(filename, 6, Fs_red);
        else
            test_set(:, (ii - 1) * 12 + (jj - 9) * 6 + (1:6)) =...
                wave_decomp(filename, 6, Fs_red);
        end
    end
```

```
end

%% SVD OF TRAINING DATA

mean_training_set = mean(training_set, 2);
[U, S, V] = svd(training_set - mean_training_set, 'econ');

% figure,
% plot(diag(S), 'ko')

% SVD Modes

[~, freq] = wave_decomp(filename, 1, Fs_red);
fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
plot_labels = {'(a)', '(b)', '(c)', '(d)'};
for mm = 1: 1: 4
    s = subplot(2, 2, mm);
    pcolor((1: Fs_red * 5) / Fs_red, freq, reshape(abs(U(:, mm)),...
        [100, Fs_red * 5]))
    shading interp
    axis tight
    colormap(pink)
    xlabel({'Time [s]'; plot_labels{mm}})
    ylabel('Frequency [Hz]')
end

% Projection of data onto principal modes

SVD_proj = (U' * training_set)';

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 6];
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 6];
plot_labels = {'(a)', '(b)', '(c)', '(d)', '(e)', '(f)'};
for mm = 1:1:6
    s1 = subplot(6, 1, mm);
    s1.Box = 'on';
    hold on
    h1 = histogram((SVD_proj(1:48, mm)), 'EdgeColor', 'r', 'FaceColor', 'r');
    h2 = histogram((SVD_proj(49:96, mm)), 'EdgeColor', 'm', 'FaceColor', 'm');
    h3 = histogram((SVD_proj(97:144, mm)), 'EdgeColor', 'b', 'FaceColor', 'b');
    h2.LineStyle = '--'; h3.LineStyle = ':';
    h1.LineWidth = 1.1; h2.LineWidth = 1.1; h3.LineWidth = 1.1;
    xlabel(plot_labels{mm})
    ylabel('Counts')
    legend('Jazz', 'Carnatic', 'Classical')
end

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
```

```
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
ax = gca;
ax.Box = 'on';
ax.View = [11.412, 63.0438];
hold on
h1 = plot3(SVD_proj(1:48, 2), SVD_proj(1:48, 3), SVD_proj(1:48, 6), 'ro');
h2 = plot3(SVD_proj(49:96, 2), SVD_proj(49:96, 3), SVD_proj(49:96, 6), 'm^');
h3 = plot3(SVD_proj(97:144, 2), SVD_proj(97:144, 3), SVD_proj(97:144, 6), 'bp');
xlabel('PCA 1'), ylabel('PCA 2'), zlabel('PCA 6')
legend('Jazz', 'Carnatic', 'Classical')

% figure;
% for mm = 1:1:6
% subplot(6, 3, (mm - 1) * 3 + 1)
% plot(1:48, (SVD_proj(1:48, mm)), 'ko-')
% subplot(6, 3, (mm - 1) * 3 + 2)
% plot(1:48, (SVD_proj(49:96, mm)), 'ko-')
% subplot(6, 3, (mm - 1) * 3 + 3)
% plot(1:48, (SVD_proj(97:144, mm)), 'ko-')
% end

%% LDA WITH CROSS VALIDATION

labels = [-1 * ones(40, 1);...
          0 * ones(40, 1);...
          1 * ones(40, 1)];
truth = [-1 * ones(8, 1);...
          0 * ones(8, 1);...
          1 * ones(8, 1)];
accuracy_lin = nan(length(truth), 1);
accuracy_quad = nan(length(truth), 1);
for nn = 1: 1: 50
    rand_vec = randperm(48);
    train_mat = [SVD_proj(rand_vec(1:40), 1:5);...
                 SVD_proj(rand_vec(1:40) + 48, 1:5);...
                 SVD_proj(rand_vec(1:40) + 96, 1:5)];
    test_mat = [SVD_proj(rand_vec(41:48), 1:5);...
                SVD_proj(rand_vec(41:48) + 48, 1:5);...
                SVD_proj(rand_vec(41:48) + 96, 1:5)];
    class_lin = classify(test_mat, train_mat, labels);
    class_quad = classify(test_mat, train_mat, labels, 'quadratic');
    accuracy_lin(nn, 1) = sum(class_lin == truth) / length(truth);
    accuracy_quad(nn, 1) = sum(class_quad == truth) / length(truth);
end

fprintf('\nTraining data:\n')
fprintf('Quadratic discriminant accuracy\n')
fprintf('Mean = %.2f \t Minimum = %.2f \t Maximum = %.2f\n',...
    mean(accuracy_quad), min(accuracy_quad), max(accuracy_quad))
fprintf('Linear discriminant accuracy\n')
fprintf('Mean = %.2f \t Minimum = %.2f \t Maximum = %.2f\n',...
    mean(accuracy_lin), min(accuracy_lin), max(accuracy_lin))

fig = figure;
fig.Units = 'inches';
fig.Position = [-.1 1.8 6.75 5.0625];
```

```
fig.PaperUnits = 'inches';
fig.PaperSize = [6.75 5.0625];
ax = gca;
ax.Box = 'on';
h1 = bar([accuracy_lin, accuracy_quad]);
legend('Linear', 'Quadratic')
xlabel('Trial')
ylabel('Accuracy')

%% LDA ON TEST DATA

test_truth = [-1 * ones(12, 1);...
              0 * ones(12, 1);...
              1 * ones(12, 1)];
test_mat =(U(:, 1:5)' * test_set)';
labels = [-1 * ones(48, 1);...
          0 * ones(48, 1);...
          1 * ones(48, 1)];
train_mat = [SVD_proj(1:48, 1:5);
             SVD_proj(49:96, 1:5);
             SVD_proj(97:144, 1:5)];
class_lin = classify(test_mat, train_mat, labels);
class_quad = classify(test_mat, train_mat, labels, 'quadratic');
accuracy_lin = sum(class_lin == test_truth) / length(test_truth);
accuracy_quad = sum(class_quad == test_truth) / length(test_truth);

fprintf('\nTest data:\n')
fprintf('Quadratic discriminant\n')
fprintf('Accuracy = %.2f\n', accuracy_quad)
fprintf('Linear discriminant\n')
fprintf('Accuracy = %.2f\n', accuracy_lin)
```

B.5 Wavelet Decomposition Function

```
%% FUNCTIION TO EXTRACT SAMPLES AND PERFORM WAVELET DECOMPOSITION

function [wavelets, freq] = wave_decomp(filename, num_of_samples, Fs_red)
    wavelets = nan(100 * 5 * Fs_red, num_of_samples);
    freq = nan(100, num_of_samples);
    info = audioinfo(filename);
    Fs = info.SampleRate;
    timing_vec = 1:5:info.Duration - 5;
    sample_times = timing_vec(randperm(length(timing_vec), num_of_samples));
    for kk = 1: num_of_samples
        sample = audioread(filename, [sample_times(kk) * Fs + 1, ...
            sample_times(kk) * Fs + 5 * Fs]);
        sample_mono = mean(sample, 2);
        resampled_mono = resample(sample_mono, Fs_red, Fs);
        [resampled_mono_wav, freq(:, kk)] = cwt(resampled_mono, 'morse',...
            Fs_red, 'WaveletParameters', [750, 750 * 40]);
        wavelets(:, kk) = abs(resampled_mono_wav(:));
    end
    freq = freq(:, 1);
end
```

Appendix C Additional figures

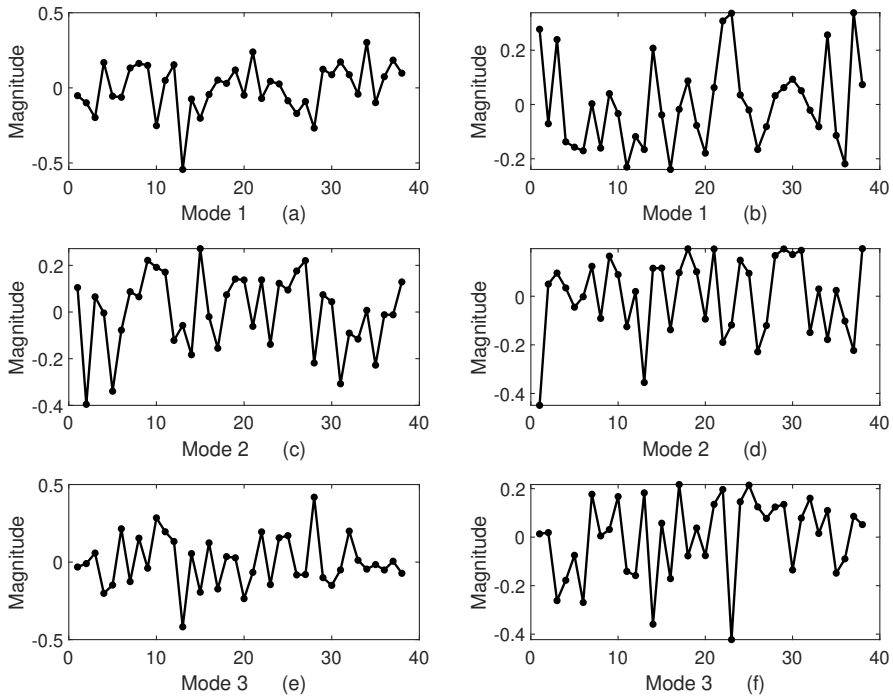


Figure 8: (a), (c), and (e) show the variations of first 3 rows of $\boldsymbol{V}*$ from SVD of cropped dataset; (b), (d), and (f) show the variations of first 3 rows of $\boldsymbol{V}*$ from SVD of uncropped dataset.

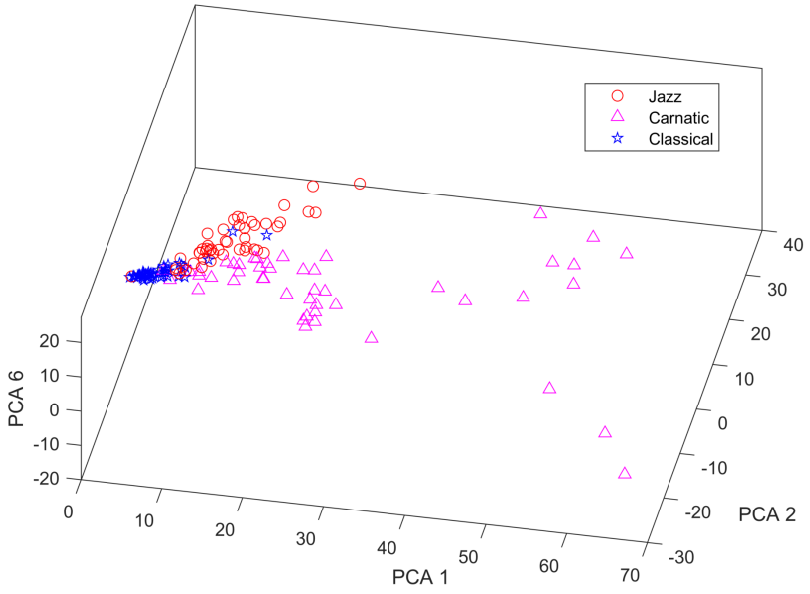


Figure 9: For Case 1, the plot shows the projection of training data onto PCA modes 1, 2, and 6, indicating how they are physically separable in space.