Homework 4: Music Classification

Joshua Yap
AMATH 482
Winter 2020

March 7, 2020

Abstract

This homework requires us to merge knowledge of PCA and LDA to construct a music classifier that classifies music by different artist and of different genres. Exploration of the parameters to control in constructing the classifier reveals potential ways to improve it and the problem of overfitting. Accuracy scores for each test are calculated and compared with plots of the data on the basis found by LDA.

1 Introduction and Overview

The problem requires us to classify music clips that are 5 seconds long by implementing a machine learning algorithm using Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). A model is trained on a training data set and then tested on another data set. In part 1, we classify music by different artists of different genres (classical, folk and electronic). In part 2, we classify music by different artists of the same genre (classical). In part 3, we distinguish music of different genres given music by multiple artists within each genre.

2 Theoretical Background

PCA is an application of Singular Value Decomposition, which was discussed in Homework 3, and will therefore only be discussed briefly here. The main idea is to optimally reduce dimensionality of the data. For a data set with 20 dimensions, SVD gives us a set of singular values and singular vectors, and PCA is the projection of data onto only the most significant components (or principal components).

2.1 Linear Discriminant Analysis

LDA allows us to best differentiate between different classes of data. For a 3-class LDA like we have, it finds a plane on which to project the data such that the distance between inter-class data is maximized while the

intra-class data is minimized. This is done by solving the generalized eigenvalue problem

$$\mathbf{S_B}\vec{w} = \lambda \mathbf{S_W}\vec{w}. \tag{1}$$

$\mathbf{S_B}$ and $\mathbf{S_W}$ are between-class and within-class scatter matrices respectively, given by

$$\mathbf{S_B} = \sum_{j=1}^{n} m_j(\vec{\mu}_j - \vec{\mu})(\vec{\mu}_j - \vec{\mu})^T, \tag{2}$$

$$\mathbf{S_W} = \sum_{j=1}^{n} \sum_{\vec{x}} (\vec{x} - \vec{\mu}_j)(\vec{x} - \vec{\mu}_j)^T. \tag{3}$$

Solving equation 1 gives us a square matrix λ with eigenvalues on the diagonal and eigenvectors in the columns of matrix $\mathbf{W}$. The columns in $\mathbf{W}$ corresponding to the 2 largest eigenvalues are the vectors that form the plane that best differentiates the data.

3 Algorithm Implementation and Development

Pre-process audio: The audio clips were converted to spectrograms using the Gabor transform.

Collect spectrograms in matrix: The spectrograms were reshaped into column vectors and put side-by-side into a matrix.

SVD: Perform Singular Value Decomposition on matrices of training data. Project original data onto first 20 or 30 principal components.

LDA: Perform Linear Discriminant Analysis to find optimal plane to project data onto. Define threshold as minimum distance to mean of a class scaled by variance of the class. A larger variance scales the distance down.

Test: Test the classifier using a separate test data set. Perform SVD and LDA on test set and use the define threshold to classify audio clips. Compare with labels to see how accurate the classifier is.

4 Computational Results

We plotted spectrograms initially to find a good filter width to use. The one chosen results, for example, in the spectrogram in figure 1.

After converting all the audio to spectrograms and performing SVD, we plotted the energy of the principal components to decide how many features to keep. Figure 2 shows the energies for 2 of the tests. Like the

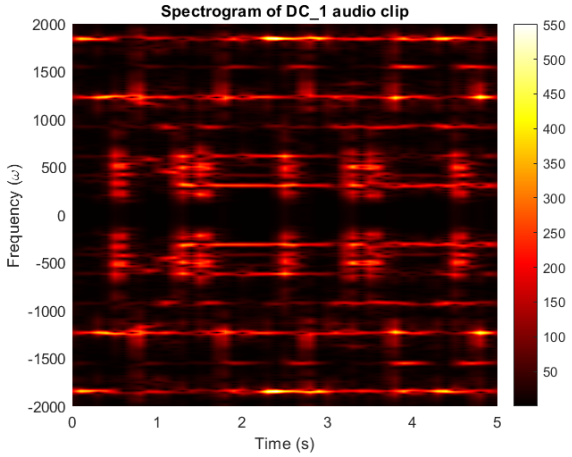
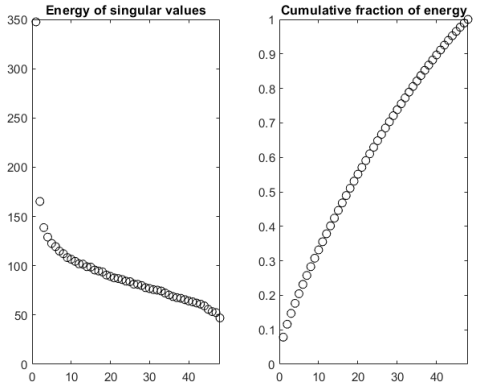
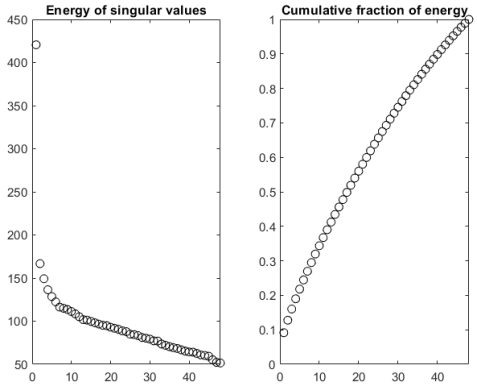


Figure 1: Parameters a and dt were manipulated to find a spectrogram with good time and frequency resolution. This is the spectrogram for a clip from test 1.



(a) Energy for test 2



(b) Energy for test 3

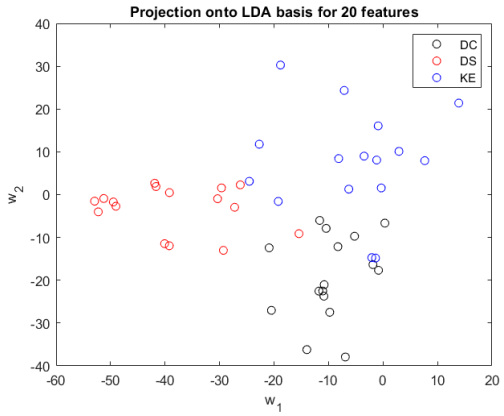
Figure 2: Energies of principal components.

dog/cat classifier in class, these have a wide tail so we decided to keep more features in our classifier.

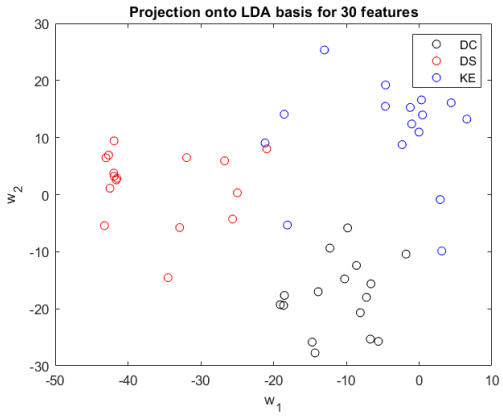
In order to verify our decision, we looked at plots of the data projected onto the plane found by LDA for different numbers of features. Figure **??** highlights the difference in separation of classes when using more features.

It looked like using 30 features would give us more accurate results on the training data set, but accuracy scores for both were computed anyway for each test. Figure 4 shows test data on top of training data. It let verify that the results were reasonable.

The accuracy score was computed based on the fraction of audio clips the classifier got correct. This is given for all tests in the table below.



(a) Projection for 20 features from test 1



(b) Projection for 30 features from test 1

Figure 3: Comparison of LDA for different numbers of features. The legends give the initials of the artists of the audio clips.

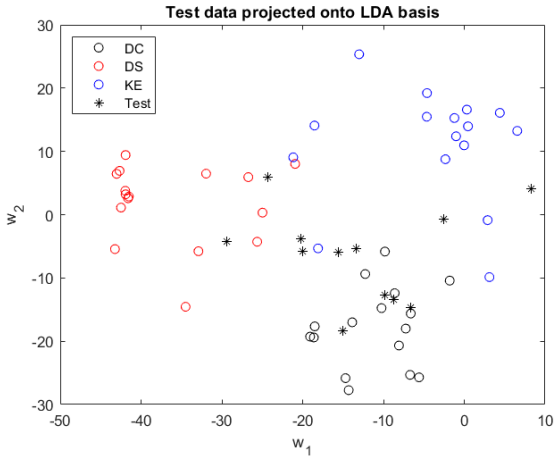
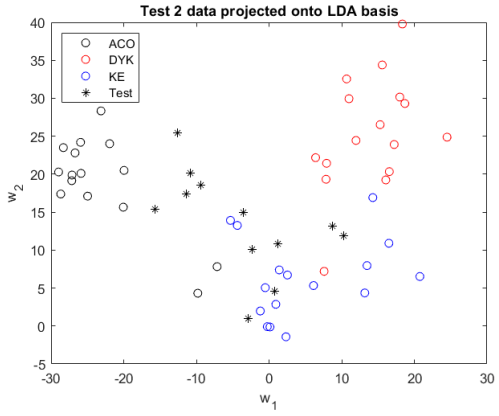
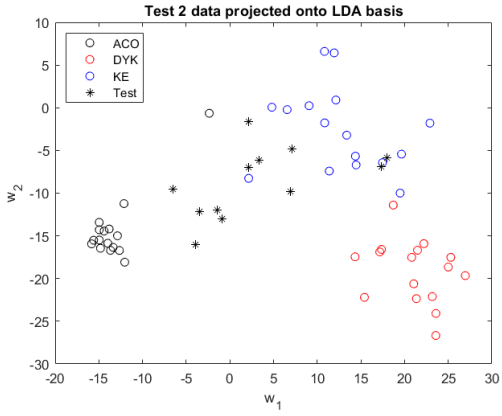


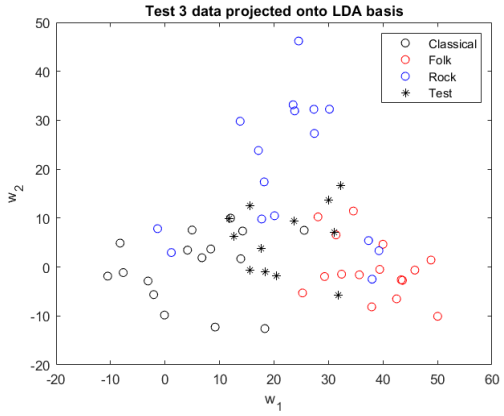
Figure 4: Test 1 results superimposed on training data.



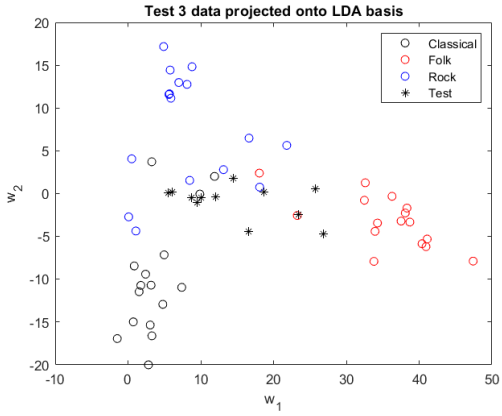
(a) Test 2, 20 features (classifying artists from the same genre)



(b) Test 2, 30 features (classifying artists from the same genre)



(c) Test 3, 20 features (classifying genres)



(d) Test 3, 30 features (classifying genres)

Figure 5: Test data plotted with training data using LDA.

Test	Features	Accuracy
1	20	0.5833
1	30	0.75
2	20	0.5833
2	30	0.33
3	20	0.4167
3	30	0.5

The other LDA plots are given in figure 5.

At this point, we can discuss our results. Comparing the table of accuracy scores with the LDA plots, it seems like the 30 feature classifiers would have performed better because it better separates the training data. However, this was not the case in test 2. On the plot, we could probably identify how the test data points should be classified, but the threshold function of scaled distance fails to classify them correctly, yielding a

dismal 0.33 success rate. To improve this, a different threshold standard can be found and tested to see if it performs better.

What also surprised me was that I used the same audio clips KE for tests 1 and 2, but the blue data points on the plots for test 1 and test 2. After thinking about it, it makes sense that they would differ since we are projecting them onto different planes caused by the difference in the other data that the KE audio clips are grouped with.

5 Summary and Conclusions

We have trained and tested a classifier to classify audio clips by artist and genre. Its performance could be improved by training it on more data (it was trained on 48 5-second clips and tested on 12 5-second clips in each test). We can also try other methods of obtaining thresholds to classify the data. Using more features may have caused the model to overfit the training data and thereby reduce performance on test data. It was therefore good that we tested the classifier on data it had not seen before.

Appendices

Appendix A

Summary of MATLAB functions used

contains: Checks if a string contains a substring, returns a boolean.

audioread: Reads in an audio file and returns a vector y of amplitudes and Fs the sampling frequency.

resample: Resamples the signal y at a new specified frequency.

var: Computes the variance of a vector or matrix along the specified dimension.

Appendix B

MATLAB code

```
close all; clear; clc
% Train classifier
[CL_data,Fs] = group_music('Music/Part_3/training/','CL');
```

```
[CL_spec,~,~,~] = audio_to_spec(CL_data,Fs);
[FK_data,Fs] = group_music('Music/Part_3/training/','FK');
[FK_spec,~,~,~] = audio_to_spec(FK_data,Fs);
[RK_data,Fs] = group_music('Music/Part_3/training/','RK');
[RK_spec,tslide,n,ks] = audio_to_spec(RK_data,Fs);
n_features = 20;
[U,S,V,CL_proj,FK_proj,RK_proj,w,cluster] = music_trainer(CL_spec,FK_spec,RK_spec,n_features);

%% Test classifier
[test3_CL,~] = group_music('Music/Part_3/testing/','CL');
[test3_FK,~] = group_music('Music/Part_3/testing/','FK');
[test3_RK,Fs] = group_music('Music/Part_3/testing/','RK');
test3_data = [test3_CL test3_FK test3_RK];
[test3_spec,tslide,n,ks] = audio_to_spec(test3_data,Fs);

test3_proj = U'*test3_spec;
pos = w'*test3_proj;

% classify test data
pred_class = zeros(1,length(pos));
for j = 1:length(pos)
    xy = pos(:,j);
    xy_diff = (xy-cluster(1:2,:))./cluster(3:4,:);
    [~,ind] = min(sum(abs(xy_diff),1));
    pred_class(j) = ind;
end

% Accuracy score
labels = [1 1 1 1 2 2 2 2 3 3 3 3];
correct = labels-pred_class; % 0 if correct
acc_score = 1-mean(correct~=0);

function [result_mat,Fs] = group_music(sub_dir,artist)
```

```
% Looks in the specified directory and groups .wav files of the
% specified type into matrices for training and testing. Subsamples
% the given audio at half the frequency.
% n_files = length(dir(sub_dir));
if contains(sub_dir,'training')
    range_file = [1:16];
else
    range_file = [17:20];
end
result_mat = zeros(110250,length(range_file));
for j = range_file
    file_name = strcat(sub_dir,artist,'_',num2str(j),'.wav');
    [y_orig,Fs_orig] = audioread(file_name);
    Fs = Fs_orig/2;
    y = resample(y_orig,Fs,Fs_orig); % Subsampling
    result_mat(:,mod(j-1,16)+1) = y;
end
end

function [music_spec,tslide,n,ks] = audio_to_spec(y_mat,Fs)
% Takes in an audio signal and returns a matrix to plot a spectrogram
[rows,cols] = size(y_mat);
n = pow2(nextpow2(rows)); % number of fourier modes
L = n/Fs;

t2 = linspace(0,L,n+1); % time discretization
t = t2(1:n);

k = (2*pi/L)*[0:n/2-1 -n/2:-1]; % frequency components
ks = fftshift(k);

% Construct Gabor window
tslide = 0:0.1:5; % where the window is centered
```

```
a = 100; % controls window width

% Initialize return matrix
music_spec = zeros(length(tslide)*n,cols);
for ii = 1:cols
    ygt_spec = zeros(n,length(tslide));
    y = y_mat(:,ii);
    for j=1:length(tslide)
        g=exp(-a*(t(1:rows)-tslide(j)).^2);
        yg=g'.*y;
        ygt=fft(yg,n);
        ygt_spec(:,j) = rescale(fftshift(abs(ygt))); % Scaled
    end
    music_spec(:,ii) = reshape(ygt_spec,length(tslide)*n,1);

end
end

function [U,S,V,v_music_1,v_music_2,v_music_3,w,clusters] = music_trainer(music_1,music_2,music_3,fe
% Takes 3 sets of music in the form of reshaped spectrograms and
% trains a model to distinguish them.
n1 = length(music_1(1,:));
n2 = length(music_2(1,:));
n3 = length(music_3(1,:));

[U,S,V] = svd([music_1 music_2 music_3],'econ');
U = U(:,1:feature);
music = S*V'; % Projection onto principal components
music_1_proj = music(1:feature,1:n1);
music_2_proj = music(1:feature,n1+1:n1+n2);
music_3_proj = music(1:feature,n1+n2+1:n1+n2+n3);

mean_1 = mean(music_1_proj,2); % column vector with mean of each row
```

```
mean_2 = mean(music_2_proj,2);
mean_3 = mean(music_3_proj,2);

Sw = 0; % Within-class variance
for j = 1:n1
    Sw = Sw + (music_1_proj(:,j)-mean_1)*(music_1_proj(:,j)-mean_1)';
end
for j = 1:n2
    Sw = Sw + (music_2_proj(:,j)-mean_2)*(music_2_proj(:,j)-mean_2)';
end
for j = 1:n3
    Sw = Sw + (music_3_proj(:,j)-mean_3)*(music_3_proj(:,j)-mean_3)';
end

mu = (mean_1+mean_2+mean_3)/3;
Sb = n1*(mean_1-mu)*(mean_1-mu)' + n2*(mean_2-mu)*(mean_2-mu)' ...
    + n3*(mean_3-mu)*(mean_3-mu)'; % Between-class variance

% Linear discriminant analysis
[V2,D] = eig(Sb,Sw);
d = diag(D);
[~,I] = sort(d,'descend'); % Find 2 highest eigenvalues
ind1 = I(1);
ind2 = I(2);
w1 = V2(:,ind1);
w2 = V2(:,ind2);
w = [w1 w2]; % Construct plane w to project data onto
w = w/norm(w,2);

v_music_1 = w'*music_1_proj;
v_music_2 = w'*music_2_proj;
v_music_3 = w'*music_3_proj;
```

```
% Determine mean and variance of clusters
mean_v1 = mean(v_music_1,2);
mean_v2 = mean(v_music_2,2);
mean_v3 = mean(v_music_3,2);
var_v1 = var(v_music_1,0,2);
var_v2 = var(v_music_2,0,2);
var_v3 = var(v_music_3,0,2);

% Collect means and variances in matrix to output
clusters = [mean_v1 mean_v2 mean_v3;
            var_v1 var_v2 var_v3];
end
```