Music Classification

Jason Miller
AMATH 482

March 6, 2020

Abstract

* Typically, we can listen to a short clip of music and quickly determine which genre the music belongs to, as well as possibly the band that performed the song, to a relatively high degree of accuracy. It turns out that we can build an algorithm that enables computers to do this as well. In this assignment we build a classifier to determine which genre and/or band a five-second clip of music belongs to.

1 Introduction and Overview

There are three parts to this project: First, we take clips from three different bands (Dee Yan-Key, Andrew Codeman, The Noontime Jamboree), each from a different genre of music, and use this as training data to build a classifier that takes in a clip from any of these bands and determines which band produced the music. Next, we take three bands from the same genre, and build a classifier that determines which band a piece of music came from. We expect this to be more difficult. Finally, we build a more general classifier that takes music that is either country, jazz, or electronic, and returns which genre of these three that it is from. We will use a machine learning algorithm called LDA to make this happen.

2 Theoretical Background

Linear Discriminant Analysis (LDA) is a method for projecting our data in such a way that maximizes distance between inter-class data while simultaneously minimizing the spread of intra-class data. That is, a projection $\mathbf{w}$ such that

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

Where equation (1) is a generalized eigenvalue problem, λ is the maximum eigenvalue, $\mathbf{w}$ is the corresponding eigenvector, and S_B and S_W are scatter matrices for between class and within class data respectively. This creates a suitable threshold with which we can seperate our data and classify each data point. Additionally, recall that the singular value decomposition is a way of decomposing a matrix into two matrices that represent rotations and one matrix that represents stretching, and a spectrogram is a way of capturing frequency information of a signal, localized to each moment in time.

3 Algorithm Implementation and Development

The process for each of the three tests is as follows:

1. Extract the song data from Free Music Archive. Truncate each song vector into a five-second clip, taken from the middle of the song (since the beginning of each song tends to sound similar). We take one song from each band for test 1, and three for each genre for test 2 and three, since a larger training set was necessary to boost accuracy.
2. Create spectrograms of each song vector so that we can take the SVD of the spectrograms later. This is because it is too computationally expensive to take the SVD of just the raw data vectors.
3. From the spectrograms, find the principal components associated with each band for test 1 and 2, or genre for test 3 respectively by concatenating the spectrogram matrices together and taking the SVD.
4. Using linear discriminant analysis, find a decision threshold to discriminate between each band, or between country, jazz, and electronic music.
5. Calculate the accuracy by testing the algorithm on 30 song snippets from each band/genre that is pre-labled.

4 Computational Results

4.1 Test One

Since each band came friom a different genre, the songs all sounded quite different from each other and therefore the distance between inter-class data was quite high. This led to an accuracy of .933, or 28 out of 30 songs.

4.2 Test Two

Here, the songs sounded more similar since they were from the same genre (jazz), and so the spread of the inter-class data was lower. This is why we needed a larger training set, and this increased our accuracy from a dismal 20 percent to a better but still unreliable 70 percent (21 out of 30 songs correctly identified). Since the songs are so audibly similar this is a reasonable result, but could likely be improved upon substantially.

4.3 Test Three

Here, since the songs in a given genre came from multiple bands, there was a larger spread of the intra-class data, which was the largest factor which the genre of the music challenging for our algorithm to predict, which is why we decided to use clips from the middle of the song. We ultimately had a 76.6 accuracy rate, correctly identifying 23 out of 30 songs.

5 Summary and Conclusions

We built a classifier to determine which band/genre of music a five-second clip belongs to using linear discriminant analysis, the singular value decomposition, and spectrograms. By inspection, it seems like the largest factor that the determinations were made based upon is which instruments were most prominant in each song. As expected, it was more difficult to obtain a higher accuracy when there was greater spread of intra-class data in our songs or lesser spread of inter-class data, since both of these conditions make it harder for the threshold obtained by LDA to effectively discriminate between the different types of music. There is room for improvement, perhaps by choosing songs for our training set that are more representative of the their respective genres and bands, but our results are quite reasonable overall.

Appendix A: Function Descriptions

- abs: 2-norm
- exp: “e to the”

- fft: Fast Fourier Transform
- fftshift: interchanges first and second halves of the domain
- initialize: initializes PointTracker
- linspace: creates vector with linearly spaced points
- load: loads .mat file
- PointTracker: constructs PointTracker object
- print: saves figure
- release: releases PointTracker
- rgb2gray: changes image to grayscale
- spectrogram: creates spectrogram
- svd: computes the SVD of a matrix
- title: titles plot
- xlabel: labels x-axis on plot
- zeros: creates matrix of zeros

Appendix B: Code

```
clear; clc; close all;

load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')

numFrames1 = size(vidFrames1_1, 4);
numFrames2 = size(vidFrames2_1, 4);
numFrames3 = size(vidFrames3_1, 4);

positionTracker = vision.PointTracker;
noiseTracker = vision.PointTracker;
```

```
x1 = zeros(1,numFrames1);
y1 = zeros(1,numFrames1);
x2 = zeros(1,numFrames2);
y2 = zeros(1,numFrames2);
x3 = zeros(1,numFrames3);
y3 = zeros(1,numFrames3);

for j = 1:numFrames1
    gray1(:,:,j) = rgb2gray(vidFrames1_1(:,:,:,j))
        ;
    if j == 1
        initialize(positionTracker, [342 292],
            gray1(:,:,1));
        initialize(noiseTracker, [20 313], gray1
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray1(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray1(:,:,j));
    x1(j) = points(1);
    y1(j) = points(2);
    %x1(j) = points(1) - (perterbation(1) - 20);
    %y1(j) = points(2) - (perterbation(2) - 313);
end

for j = 1:numFrames2
    gray2(:,:,j) = rgb2gray(vidFrames2_1(:,:,:,j))
        ;
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [283 344],
            gray2(:,:,1));
        initialize(noiseTracker, [58 175], gray2
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray2(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray2(:,:,j));
    x2(j) = points(1);
```

```matlab
    y2(j) = points(2);
    %x2(j) = points(1) - (perterbation(1) - 58);
    %y2(j) = points(2) - (perterbation(2) - 175);
end

for j = 1:numFrames3
    gray3(:,:,j) = rgb2gray(vidFrames3_1(:,:,:,j));
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [366 288], gray3(:,:,1));
        initialize(noiseTracker, [554 75], gray3(:,:,1));
    end
    [points, pointValidity] = positionTracker(gray3(:,:,j));
    [perterbation, pertValidity] = noiseTracker(gray3(:,:,j));
    x3(j) = points(1);
    y3(j) = points(2);
    %x3(j) = points(1) - (perterbation(1) - 58);
    %y3(j) = points(2) - (perterbation(2) - 175);
end
temp = x3;
x3 = y3;
y3 = temp;

start1 = 10;
x1 = x1(start1 : start1 + 199);
y1 = y1(start1 : start1 + 199);

start2 = 9;
x2 = x2(start2 : start2 + 199);
y2 = y2(start2 : start2 + 199);

start3 = 10;
x3 = x3(start3 : start3 + 199);
y3 = y3(start3 : start3 + 199);

P = [x1;y1;x2;y2;x3;y3];
[U,S,V] = svd(P, 'econ');
```

```
svs = diag(S);
e1 = svs(1) / sum(svs)
```

```
clear; clc; close all;

load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')

numFrames1 = size(vidFrames1_2, 4);
numFrames2 = size(vidFrames2_2, 4);
numFrames3 = size(vidFrames3_2, 4);

positionTracker = vision.PointTracker;
noiseTracker = vision.PointTracker;

x1 = zeros(1,numFrames1);
y1 = zeros(1,numFrames1);
x2 = zeros(1,numFrames2);
y2 = zeros(1,numFrames2);
x3 = zeros(1,numFrames3);
y3 = zeros(1,numFrames3);

%Initializing these breaks the code for some
    reason
%gray1 = zeros(480,640,numFrames1);
%gray2 = zeros(480,640,numFrames2);
%gray3 = zeros(480,640,numFrames3);

for j = 1:numFrames1
    gray1(:,:,j) = rgb2gray(vidFrames1_2(:,:,:,j))
        ;
    if j == 1
        initialize(positionTracker, [323 358],
            gray1(:,:,1));
        initialize(noiseTracker, [44 71], gray1
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray1(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray1(:,:,j));
    x1(j) = points(1);
    y1(j) = points(2);
    %x1(j) = points(1) - (perterbation(1) - 20);
    %y1(j) = points(2) - (perterbation(2) - 313);
end

for j = 1:numFrames2
    gray2(:,:,j) = rgb2gray(vidFrames2_2(:,:,:,j))
        ;
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [312 368],
            gray2(:,:,1));
        initialize(noiseTracker, [58 99], gray2
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray2(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray2(:,:,j));
    %x2(j) = points(1);
    %y2(j) = points(2);
    x2(j) = points(1) - (perterbation(1) - 58);
    y2(j) = points(2) - (perterbation(2) - 175);
end

for j = 1:numFrames3
    gray3(:,:,j) = rgb2gray(vidFrames3_2(:,:,:,j))
        ;
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [383 271],
            gray3(:,:,1));
        initialize(noiseTracker, [20 20], gray3
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(gray3(:,:,j));
    [perterbation, pertValidity] = noiseTracker(gray3(:,:,j));
    x3(j) = points(1);
    y3(j) = points(2);
    %x3(j) = points(1) - (perterbation(1) - 58);
    %y3(j) = points(2) - (perterbation(2) - 175);
end
temp = x3;
x3 = y3;
y3 = temp;

start1 = 4;
x1 = x1(start1 : start1 + 199);
y1 = y1(start1 : start1 + 199);

start2 = 8;
x2 = x2(start2 : start2 + 199);
y2 = y2(start2 : start2 + 199);

start3 = 8;
x3 = x3(start3 : start3 + 199);
y3 = y3(start3 : start3 + 199);

P = [x1;y1;x2;y2;x3;y3];
[U,S,V] = svd(P, 'econ');
svs = diag(S);
e1 = svs(1) / sum(svs)
```

```
clear; clc; close all;

load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')

numFrames1 = size(vidFrames1_3, 4);
numFrames2 = size(vidFrames2_3, 4);
numFrames3 = size(vidFrames3_3, 4);

positionTracker = vision.PointTracker;
noiseTracker = vision.PointTracker;

x1 = zeros(1,numFrames1);
y1 = zeros(1,numFrames1);
x2 = zeros(1,numFrames2);
y2 = zeros(1,numFrames2);
x3 = zeros(1,numFrames3);
y3 = zeros(1,numFrames3);

%Initializing these breaks the code for some
    reason
%gray1 = zeros(480,640,numFrames1);
%gray2 = zeros(480,640,numFrames2);
%gray3 = zeros(480,640,numFrames3);

for j = 1:numFrames1
    gray1(:,:,j) = rgb2gray(vidFrames1_3(:,:,:,j))
        ;
    if j == 1
        initialize(positionTracker, [340 340],
            gray1(:,:,1));
        initialize(noiseTracker, [20 20], gray1
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray1(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray1(:,:,j));
    x1(j) = points(1);
    y1(j) = points(2);
    %x1(j) = points(1) - (perterbation(1) - 20);
    %y1(j) = points(2) - (perterbation(2) - 313);
end

for j = 1:numFrames2
```

```
    gray2(:,:,j) = rgb2gray(vidFrames2_3(:,:,:,j))
        ;
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [254 352],
            gray2(:,:,1));
        initialize(noiseTracker, [20 20], gray2
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray2(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray2(:,:,j));
    x2(j) = points(1);
    y2(j) = points(2);
    %x2(j) = points(1) - (perterbation(1) - 58);
    %y2(j) = points(2) - (perterbation(2) - 175);
end

for j = 1:numFrames3
    gray3(:,:,j) = rgb2gray(vidFrames3_3(:,:,:,j))
        ;
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [405 242],
            gray3(:,:,1));
        initialize(noiseTracker, [20 20], gray3
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray3(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray3(:,:,j));
    x3(j) = points(1);
    y3(j) = points(2);
    %x3(j) = points(1) - (perterbation(1) - 58);
    %y3(j) = points(2) - (perterbation(2) - 175);
end
temp = x3;
x3 = y3;
y3 = temp;
```

```

start1 = 10;
x1 = x1(start1 : start1 + 199);
y1 = y1(start1 : start1 + 199);

start2 = 9;
x2 = x2(start2 : start2 + 199);
y2 = y2(start2 : start2 + 199);

start3 = 10;
x3 = x3(start3 : start3 + 199);
y3 = y3(start3 : start3 + 199);

P = [x1;y1;x2;y2;x3;y3];
[U,S,V] = svd(P, 'econ');
svs = diag(S);
e1 = svs(1) / sum(svs)
```

```
clear; clc; close all;

load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')

numFrames1 = size(vidFrames1_4, 4);
numFrames2 = size(vidFrames2_4, 4);
numFrames3 = size(vidFrames3_4, 4);

positionTracker = vision.PointTracker;
noiseTracker = vision.PointTracker;

x1 = zeros(1,numFrames1);
y1 = zeros(1,numFrames1);
x2 = zeros(1,numFrames2);
y2 = zeros(1,numFrames2);
```

```
x3 = zeros(1,numFrames3);
y3 = zeros(1,numFrames3);

%Initializing these breaks the code for some reason
%gray1 = zeros(480,640,numFrames1);
%gray2 = zeros(480,640,numFrames2);
%gray3 = zeros(480,640,numFrames3);

for j = 1:numFrames1
    gray1(:,:,j) = rgb2gray(vidFrames1_4(:,:,:,j));
    if j == 1
        initialize(positionTracker, [381 317], gray1(:,:,1));
        initialize(noiseTracker, [20 20], gray1(:,:,1));
    end
    [points, pointValidity] = positionTracker(gray1(:,:,j));
    [perterbation, pertValidity] = noiseTracker(gray1(:,:,j));
    x1(j) = points(1);
    y1(j) = points(2);
    %x1(j) = points(1) - (perterbation(1) - 20);
    %y1(j) = points(2) - (perterbation(2) - 313);
end

for j = 1:numFrames2
    gray2(:,:,j) = rgb2gray(vidFrames2_4(:,:,:,j));
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [256 289], gray2(:,:,1));
        initialize(noiseTracker, [20 20], gray2(:,:,1));
    end
    [points, pointValidity] = positionTracker(gray2(:,:,j));
    [perterbation, pertValidity] = noiseTracker(gray2(:,:,j));
    x2(j) = points(1);
    y2(j) = points(2);
    %x2(j) = points(1) - (perterbation(1) - 58);
    %y2(j) = points(2) - (perterbation(2) - 175);
end

for j = 1:numFrames3
    gray3(:,:,j) = rgb2gray(vidFrames3_4(:,:,:,j))
        ;
    if j == 1
        release(positionTracker);
        release(noiseTracker);
        initialize(positionTracker, [400 210],
            gray3(:,:,1));
        initialize(noiseTracker, [20 20], gray3
            (:,:,1));
    end
    [points, pointValidity] = positionTracker(
        gray3(:,:,j));
    [perterbation, pertValidity] = noiseTracker(
        gray3(:,:,j));
    x3(j) = points(1);
    y3(j) = points(2);
    %x3(j) = points(1) - (perterbation(1) - 58);
    %y3(j) = points(2) - (perterbation(2) - 175);
end
temp = x3;
x3 = y3;
y3 = temp;

start1 = 10;
x1 = x1(start1 : start1 + 199);
y1 = y1(start1 : start1 + 199);

start2 = 9;
x2 = x2(start2 : start2 + 199);
y2 = y2(start2 : start2 + 199);

start3 = 10;
x3 = x3(start3 : start3 + 199);
y3 = y3(start3 : start3 + 199);

P = [x1;y1;x2;y2;x3;y3];
```

```
[U,S,V] = svd(P, 'econ');
svs = diag(S);
e1 = svs(1) / sum(svs)

plot(y1);
ylabel('Y Coordinate');
xlabel('Time');
title('Scenario 3, Camera 1 Y-Component');
print(gcf, '-dpng','3y.png');
```