Music Classification:

I Walk the Line(ar Discriminant Analysis)

Tim Ferrin

March 6, 2020

Abstract

Humans are able to listen to songs and, given enough familiarity and exposure, classify those songs by artist and genre with near perfect success. How is it that you can hear a song on the radio and instantly know that it's a country song? This report aims to complete a group of similar tasks using Linear Discriminant Analysis.

I Introduction and Overview

How are humans able to almost instantly recognize artists and genres of songs based on the way they sound? Years and years of information gathering as members of society allow us to know generally what components make a country song or what instruments are typically heard in a classical piece of music. Computers however, do not have the luxury of being able to spend years in our society learning about what makes up a particular genre. What they can do, is add and multiply numbers really quickly (among other, more complicated tasks). Computers also cannot "hear" in the classical sense, so we will have to help them in interpreting what sounds a song makes.

The mathematical code detailed at the end of this report aims to do three things:

1. Classify 5 second clips of songs by three different artists in different genres
2. Classify 5 second clips of songs by three different artists in the same genre
3. Classify 5 second clips of songs from three different genres

First, various songs will be recorded by the computer itself using Audacity, then clipped and exported to a WAV file. Because this report involves using copyrighted music, I cannot post the music files with this report. For the first task, the three artists used are Bob Dylan, Billy Joel, and Wolfgang Amadeus Mozart (respectively hailing from the folk, rock, and classical genres). For the second task, the three artists used are Bob Dylan, Cat Stevens, and Jim Croce, all of whom primarily would be thought of as in the folk genre. If the reader isn't familiar with many songs by these performers, I would highly recommend that they add

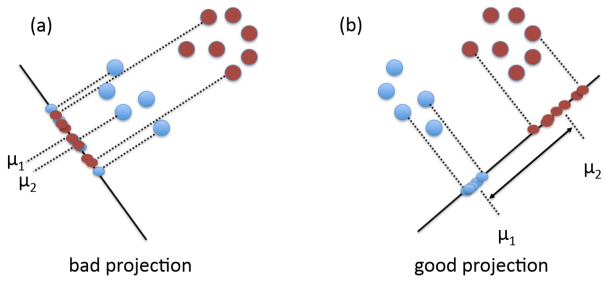

Figure 1: An example of a bad and good projection. In the left image, when the data sets are projected onto the vector, they are indistinguishable. In the right image, the data sets are clearly separated and a threshold between the two means can be determined. This image is taken from page 456 of J. Nathan Kutz's *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*

them to their Spotify playlist. For the third task, the genres of classical, folk, and pop are used.

Although many different mathematical tools for classification have been developed, the tool used for this report will be Linear Discriminant Analysis (LDA), as described in the next section.

II Theoretical Background

A prior understanding of Fourier analysis/spectrograms and of Singular Value Decomposition (SVD) is assumed for the reading of this report. Below, the concept of LDA will be explained.

The goal of LDA is simple enough to describe: Given a set of data vectors, project those vectors onto a new, useful basis vector. What does useful mean in this context? It means that the two or more projected data vectors should be easily split apart such that the means of the sets are as far apart as possible and the variances within each category are as small as possible. Figure 1 shows an example of a good projection and a bad projection.

Mathematically, this description means that we would like to find our projection vector $\mathbf{w}$ such that

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{1}$$

where the respective between-class $\mathbf{S}_B$ and within-class $\mathbf{S}_W$ scatter matrices (assuming you have the same number of data vectors/clips for each category) are

$$\mathbf{S}_B = \sum_{j=1}^{n} (\mu_j - \mu)(\mu_j - \mu)^T \tag{2}$$

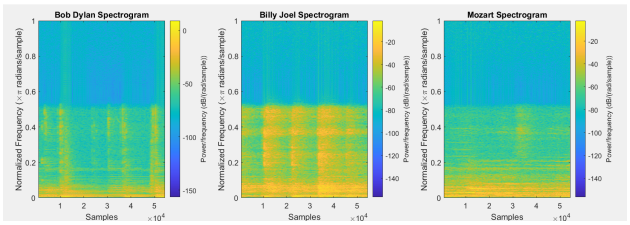

Figure 2: Spectrograms of various 5 second clips of Bob Dylan's, Billy Joel's, and Mozart's music.

$$\mathbf{S}_W = \sum_{j=1}^{n} \sum_{\mathbf{x}} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T \tag{3}$$

In equations (2) and (3), n corresponds to the number of categories you are distinguishing between, μ_j is the mean vector whose elements are the row means of all the data column vectors lined up in a matrix for each category, and μ is the mean vector whose elements are the row means of all data column vectors lined up for all categories.

Equation (1) leads to the generalized eigenvalue problem of

$$\mathbf{S}_B\mathbf{w} = \lambda \mathbf{S}_W\mathbf{w} \tag{4}$$

where the eigenvector corresponding to the maximum eigenvalue in equation (4) is the desired projection basis.

III Algorithm Implementation and Development

Following along the main code in Appendix B, the algorithm for classification (in regards to the first case specifically, although it is the exact same for each case) is as follows:

The required audio files are are loaded into the workspace and concatenated together if needed. Altogther, each audio file for each artist is about an hour long, the reason for such length is specified later. The audio data for each artist is then processed in the process() function. This function first takes the data and extracts every fourth point to reduce computational speeds. Then, the specified number of 5 second clips are selected from random points within the vectors. This is to ensure a random distribution of clips and to create a diverse training set. The spectrograms (see figure 2) of this data is then unraveled and stored in the columns of a data matrix. All recording of the audio was done in mono and at 44,100 samples per second.

The three data matrices obtained this way are then fed into the trainer() function which performs the LDA. As described in the Theoretical Background section, the various mean vectors and scatter matrices are computed. After that, the $\mathbf{w}$ projection vector is determined

and the data vectors for each category matrix are projected onto **w**. The threshold value between two categories is determined to be the point at which the same number of respective standard deviations away from the respective means occurs. For example, given the data sets projected onto the **w** with means and standard deviations μ_1,σ_1 and μ_2,σ_2, the threshold is determined by

$$\begin{aligned} \mu_1 + a\sigma_1 &= \mu_2 - a\sigma_2 \\ a\sigma_1 + a\sigma_2 &= \mu_2 - \mu_1 \\ a(\sigma_1 + \sigma_2) &= \mu_2 - \mu_1 \\ a &= \frac{\mu_1 - \mu_2}{\sigma_1 + \sigma_2} \end{aligned}$$

$$\text{Threshold} = \mu_1 + a\sigma_1 \tag{5}$$

Below this method is a commented out, alternate way to determine thresholds, but these were found to have worse classification success rates, so this method was not used. After finding the thresholds, the testing data vectors (having previously been processed) are first projected along the U^{-1} basis for the PCA projection and then projected along the **w** vector. The scalar values that this second projection spits out are then compared with to the thresholds to determine their classification. Since it is already known what category each set of test data belongs to, these known labels are compared with the LDA labels and a success rate is determined.

IV Computational Results

For the first task, when testing 50 5 second clips of Bob Dylan, Billy Joel, and Mozart songs, 54% 42%, and 86% respectively were correctly identified. However, 88% of Bob Dylan's songs were classified as either his own or Billy Joel's and 84% of Billy Joel's songs were classified as either his own or Bob Dylan's. This suggests that, at least for the clips taken from their songs recorded, that they are very similar. This occurred when using 160 features, due to the fact that it took a very long time for the singular value energies to decrease in magnitude (see figure 3). For the second task, the respective percentages were 65%, 72%, and 68% for Bob Dylan, Cat Stevens, and Jim Croce. For the third and final task, the respective percentages were 92%, 90%, and 84% for folk, classical and pop genres respectively.

V Summary and Conclusions

As seen from the computational results for task 1, and task 2, if artists have similar sounding music, it can often be morre difficult for the algorithm to distinguish them than if the artists sound totally different. It can also sometimes be difficult for humans to distinguish them,

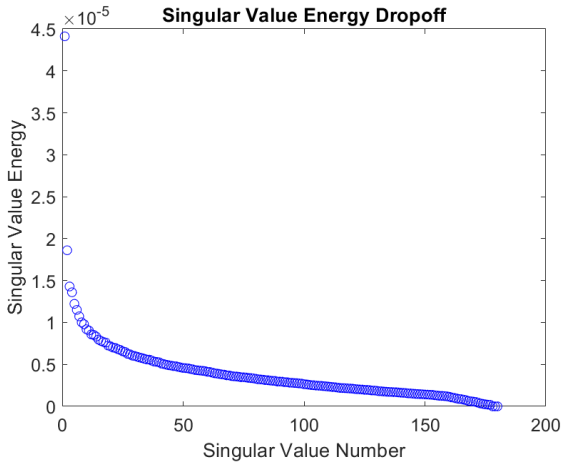

Figure 3: Dropoff of the energies for the singular values of the first task separating Bob Dylan, Billy Joel, and Mozart music. Even though the energy per singular value appears to drop off rather quickly, a large number of features must be used for the LDA.

given the number of times I've played a Billy Joel song for my friends and they've asked if the person they heard singing was Elton John. Additionally, the success rate determined by the algorithm is very much dependent on the training data available. If this report were to be expanded upon, a larger number of 5 second clips from a larger variety of songs would be used to train each set. Furthermore, the success of the testing data depended highly on whether or not the song that it came from was "typical" for that artist. As is often the case with musicians and genres, certain songs will simply be experimental and difficult to classify. This fact is acceptable, as no algorithm is ever going to be 100% correct, even the algorithm of human intuition. Overall, LDA provides a powerful tool that can handle large data sets, although not exceedingly large in MATLAB, and parse them into meaningful categories. This is highly applicable to many real world problem sets.

Appendix A: MATLAB Functions Used and Brief Implementation Explanation

audioread(file) − Converts an audio file into a usable vector of data.

spectrogram(data,width,overlap,'yaxis') − Displays the spectrogram with frequencies on the y-axis, using windows of the specified width and overlap.

find(number) − Finds the index of the desired number

randi(number) − Gives a uniformly distributed random integer between 1 and the specified number

svd(data,'econ') − Used to decompose the data matrix into its SVD matrices.

plot(x,y) – Plots the x vs y data

diag(A) – Returns the diagonal elements of the specified matrix A

mean(vector)/std(vector) – Returns the mean/standard deviation of the specified vector

sort(vector) – Sorts the specified vector

Appendix B: MATLAB Codes

For tasks 2 and 3, all individual name references were simply replaced with the musician/-genre counterpart.

```
%% Part 1)
% Load audio files
clear all;clc
[yBob1,~] = audioread('Bob1.wav');
[yBob2,~] = audioread('Bob2.wav');
[yBob3,~] = audioread('Bob3.wav');
[yBilly1,~] = audioread('Billy1.wav');
[yBilly2,~] = audioread('Billy2.wav');
[yBilly3,~] = audioread('Billy3.wav');
[yMozart1,~] = audioread('Mozart1.wav');
[yMozart2,~] = audioread('Mozart2.wav');
[yMozart3,~] = audioread('Mozart3.wav');

%% Processing audio clips and their spectrograms

yBob = [yBob1; yBob2];
yBilly = [yBilly1; yBilly2];
yMozart = [yMozart1; yMozart2];

trainNum = 60;
BobData = process(yBob,trainNum);
BillyData = process(yBilly,trainNum);
MozartData = process(yMozart,trainNum);

testNum = 50;
TestBob = process(yBob3,testNum);
TestBilly = process(yBilly3,testNum);
TestMozart = process(yMozart3,testNum);

%% Sample Spectrograms of the music
figure(1)
subplot(1,3,1)
```

```
audio = reshape(yBob,length(yBob)/4,4);
audio = audio(:,1);
Fs = 44100;
Fsnew = Fs/4;
i = 20;
spectrogram(audio((i*Fsnew):((i+5)*Fsnew)-1),1500,1400,'yaxis')
set(gca,'Fontsize',12)
title('Bob Dylan Spectrogram')
print(gcf,'BobSpect.png','-dpng')
subplot(1,3,2)
audio = reshape(yBilly,length(yBilly)/4,4);
audio = audio(:,1);
Fs = 44100;
Fsnew = Fs/4;
i = 20;
spectrogram(audio((i*Fsnew):((i+5)*Fsnew)-1),1500,1400,'yaxis')
set(gca,'Fontsize',12)
title('Billy Joel Spectrogram')
print(gcf,'BillySpect.png','-dpng')
subplot(1,3,3)
audio = reshape(yMozart,length(yMozart)/4,4);
audio = audio(:,1);
Fs = 44100;
Fsnew = Fs/4;
i = 30;
spectrogram(audio((i*Fsnew):((i+5)*Fsnew)-1),1500,1400,'yaxis')
set(gca,'Fontsize',12)
title('Mozart Spectrogram')
print(gcf,'MozartSpect.png','-dpng')
%% LDA

feature = 160;

[U,~,~,w,~,~,~,thresh1,thresh2,order] = trainer(BobData,BillyData,
    MozartData,feature);

%% Testing new song clips

valuesBob = w'*U'*TestBob;
valuesBilly = w'*U'*TestBilly;
valuesMozart = w'*U'*TestMozart;

nameVec = ["Bob" "Billy" "Mozart"];
[nameVec(order);"1" "2" "3"]

```

```
idsBob = 1+sum([valuesBob>thresh1;valuesBob>thresh2]);
idsBilly = 1+sum([valuesBilly>thresh1;valuesBilly>thresh2]);
idsMozart = 1+sum([valuesMozart>thresh1;valuesMozart>thresh2]);

percentBob = (sum(idsBob == find(order==1)))/testNum
percentBilly = (sum(idsBilly == find(order==2)))/testNum
percentMozart = (sum(idsMozart == find(order==3)))/testNum


%%
testNum = 60;
valuesBob = w'*U'*BobData;
valuesBilly = w'*U'*BillyData;
valuesMozart = w'*U'*MozartData;

nameVec = ["Bob" "Billy" "Mozart"];
[nameVec(order);"1" "2" "3"]

idsBob = 1+sum([valuesBob>thresh1;valuesBob>thresh2]);
idsBilly = 1+sum([valuesBilly>thresh1;valuesBilly>thresh2]);
idsMozart = 1+sum([valuesMozart>thresh1;valuesMozart>thresh2]);

percentBob = (sum(idsBob == find(order==1)))/trainNum
percentBilly = (sum(idsBilly == find(order==2)))/trainNum
percentMozart = (sum(idsMozart == find(order==3)))/trainNum

function data = process(audio,numFrames)
    Fs = 44100;
    Fsnew = Fs/4;
    audio = reshape(audio,length(audio)/4,4);
    audio = audio(:,1);

    fiver = audio(Fsnew:(6*Fsnew-1));
    Sp = spectrogram(fiver,1500,1400,'yaxis');
    [m,n] = size(Sp);

    data = zeros(m*n,numFrames);
    for frame = 1:numFrames
        i = randi(floor(length(audio)/Fsnew)-5);
        fiver = audio((i*Fsnew):((i+5)*Fsnew)-1);
        Sp = spectrogram(fiver,1500,1400,'yaxis');
        [m,n] = size(Sp);
        data(:,frame) = reshape(abs(Sp),m*n,1);
    end
end
```

```
122
123 function [U,S,V,w,proj1,proj2,proj3,thresh1,thresh2,order] =
    trainer(data1,data2,data3,feature)
124     n1 = size(data1,2);
125     n2 = size(data2,2);
126     n3 = size(data3,2);
127
128     [U,S,V] = svd([data1 data2 data3],'econ');
129
130     clips = S*V'; % projection onto principal components
131     U = U(:,1:feature);
132     data1s = clips(1:feature,1:n1);
133     data2s = clips(1:feature,n1+1:n1+n2);
134     data3s = clips(1:feature,n1+n2+1:n1+n2+n3);
135
136     figure(2)
137     plot(1:length(diag(S)),(diag(S))/sum(diag(S).^2),'bo')
138     set(gca,'Fontsize',12)
139     xlabel('Singular Value Number')
140     ylabel('Singular Value Energy')
141     title('Singular Value Energy Dropoff')
142     print(gcf,'SVDEnergies.png','-dpng')
143
144     m1 = mean(data1s,2);
145     m2 = mean(data2s,2);
146     m3 = mean(data3s,2);
147     m = mean([m1 m2 m3]);
148
149     Sw = 0; % within class variances
150     for k = 1:n1
151         Sw = Sw + (data1s(:,k)-m1)*(data1s(:,k)-m1)';
152     end
153     for k = 1:n2
154         Sw = Sw + (data2s(:,k)-m2)*(data2s(:,k)-m2)';
155     end
156     for k = 1:n3
157         Sw = Sw + (data3s(:,k)-m3)*(data3s(:,k)-m3)';
158     end
159
160     Sb = (m1-m)*(m1-m)'+(m2-m)*(m2-m)'+(m3-m)*(m3-m)';
161     % between class variances
162
163     [V2,D] = eig(Sb,Sw);
164     [~,ind] = max(abs(diag(D)));
165     w = V2(:,ind); w = w/norm(w,2);
```

```
166
167     proj1 = w'*data1s; proj2 = w'*data2s; proj3 = w'*data3s;
168
169     mProj1 = mean(proj1); mProj2 = mean(proj2); mProj3 = mean(
            proj3);
170     stdProj1 = std(proj1); stdProj2 = std(proj2); stdProj3 = std(
            proj3);
171
172     meanVec = [mProj1 mProj2 mProj3];
173     stdVec = [stdProj1 stdProj2 stdProj3];
174     [~,order] = sort(meanVec);
175     a1 = (meanVec(order(2))-meanVec(order(1)))/(stdVec(order(1))+
            stdVec(order(2)));
176     a2 = (meanVec(order(3))-meanVec(order(2)))/(stdVec(order(2))+
            stdVec(order(3)));
177     thresh1 = meanVec(order(1))+a1*stdVec(order(1));
178     thresh2 = meanVec(order(2))+a2*stdVec(order(2));
179
180     %sortVec = [sort(proj1);sort(proj2);sort(proj3)];
181     %low = sortVec(order(1),:);
182     %mid = sortVec(order(2),:);
183     %high = sortVec(order(3),:);
184     %t1 = length(low);
185     %t2 = 1;
186     %while low(t1)>mid(t2)
187         %t1 = t1-1;
188         %t2 = t2+1;
189     %end
190     %thresh1 = (low(t1)+low(t2))/2;
191     %t1 = length(mid);
192     %t2 = 1;
193     %while mid(t1)>high(t2)
194         %t1 = t1-1;
195         %t2 = t2+1;
196     %end
197     %thresh2 = (mid(t1)+high(t2))/2;
198 end
```