AMATH 482 Winter 2020

Homework 4: Music Classification

Tanner Graves

Introduction and Overview

Abstract

Despite musical information being something that is so naturally interpreted by us, the differences that separate musical works is something that is not as intuitively quantified. Here we will utilize methods of signal processing, statistics, and linear algebra to classify the difference between musical works. Classification will be explored on three levels: between any three artists, three artists within the same genre, and between works in three different genres.

Though we can easily recognize difference between songs, artists, and genres, the displacement over time information of a speaker creating music couldn't be further removed from the stylistic properties like pitch, rhythm, tempo, phrasing and timbre that vary so wildly between music. These musical can be more easily thought of as a relationship of what frequencies are present in a signal at a given time. For this reason, we will utilize spectrograms extensively to get a handle on this information. However, finding patterns between spectrograms remains non-trivial. Principal Component Analysis(PCA) is an incredibly useful tool for representing these complicated relationships in a way that best highlights differences and will be used in conjunction Linear Discriminant Analysis(LDA) to make some claim as to what should actually be classified as what.

The process of creating a classifier follows a process of training an algorithm with a set of dedicated training data and a set of test data that remains unseen during the training phase.

Theoretical Background

Statistical analysis is done on spectrograms of the audio by means of SVD. This process results in matrices U, Σ, and V. Where U and V form orthonormal bases and the entries in diagonal matrix Σ correspond to the variances of the original data projected onto its corresponding column of U known as its principal component. The practical use of this in this context is that if we take the principal component corresponding to the largest singular values, that is, the directions in which the data has the most variance we will best capture the characteristics necessary for the construction of a profile of a set of sounds. Approximations with a limited number of columns of U are compiled. Only using a fraction of the modes is important to avoid issues with over fitting; the integrity of overall patterns is more important that the accuracy of individual classifications.

With a good idea of what patterns are present, the matter at the heart of classification may be considered. Linear discriminant analysis aims to find a projection of a set of data that best represents variation between groups while trying to mitigate variation within groups. The vector that achieves this given two classes is described formally as follows:

$$w = \arg\max_{w} \frac{w^T S_B w}{w^T S_W w} \qquad \textit{Equation 1}$$

Where

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \qquad \textit{Equation 2}$$

$$S_W = \sum_{j=1}^{2} \sum_{x} (x - \mu_j)(x - \mu_j)^T$$ *Equation 3*

Solutions to equation one may be interpreted as the following eigenvalue problem readily solved in MATLAB:

$$S_B w = \lambda S_W w$$ *Equation 4*

After projecting, there are a few methods for determining a threshold value. However, simplest is to set the threshold between the two points that ensures that one gets equally many false positives as negatives.

Algorithm Implementation and Development

After finding enough suitable music in my parents CD collection, the first task is was to cut up the data into 5 second clips. After some experimentation, 80 training clips and 20 test clips seemed to give perfectly satisfactory classification results. This greatly reduced the amount of data I was required to process. I would only need 500 seconds of data per group, and since I was processing hours of data at a time, I could uniformly sample all songs at different points in each song to get the required 100 samples. Furthermore, once I generated the spectrogram for each clip, about the highest 80% of frequency data was found not to contain much significant audio information, so it was discarded. Since audio is sampled at such a high rate, the number of columns of each spectrogram was typically 1024, whereas most musical information relevant on a compositional level will seldom change faster than one sixteenth of a second, so I could down sample this to result in 100 by 100 spectrograms with a total element count of 10,000 that played nicely with SVD at 2% their original size.

Little processing needed to be done to the time-frequency information itself within each spectrogram. Unlike photos of physical objects, spectrograms are quite direct in the information it portrays. A value of a pixel in the spectrogram corresponds to the presence of a frequency in the original signal. As a result, it was unnecessary Discrete Waveform Transfer or other types of edge detection.

Significant thought was invested into how best to go about classification of 3 different groups. Classification will ultimately be determined by a set of subspaces that best describes the training data. In the simplest case, two groups will be projected via LDA onto the space described by vector w given by PCA. Projecting data from additional groups onto w would be convenient for the sake of classification, as it is very easy to find a threshold the divide the data into additional subspaces in 1D. However, the optimality of w for projection guaranteed by PCA is lost with any further projection. Instead of projecting all groups onto one line or a plane, I noted that by classification between every two of the three groups one pair at a time provides an optimal basis for classification between groups. For three groups A, B, and C we will have a dimension where the difference between A and B is greatest, and similarly where the A-C and B-C differences are greatest for the other two.

From this perspective, classification may become a subtly different question, instead of dividing space, we may instead ask how confident any given classification is and weigh all classifications involving a particular group. With two groups, classification is Boolean and determined by:

$$PV > t$$ *Equation 5*

Where PVs are the p-values given by LDA and t's are thresholds. With several, the result is a vector with each component being a sum of distances from threshold in the favor of a respective group.

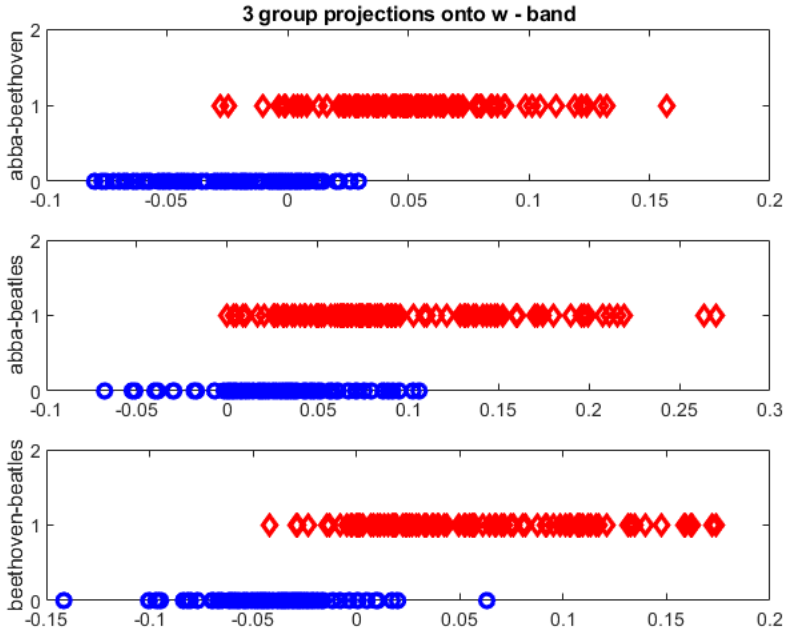
$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} -(PV_{ab} - t_{ab}) - (PV_{ac} - t_{ac}) \\ PV_{ab} - t_{ab} - (PV_{bc} - t_{bc}) \\ PV_{bc} - t_{bc} + PV_{ac} - t_{ac} \end{pmatrix}$$ *Equation 6*

Once this vector is obtained, the classification comes down to seeing what component has the maximum value. For example: the vector (0.1122, -0.0721, -0.0401) would classify the point as a member of group A.

Computational Results

Part I: Inter-artist, inter-genre classification

Selections of ABBA, the Beatles, and Beethoven proved to be disparate enough to make the task of classification pretty straightforward. The success rate of classification is 80%, as a result of satisfactory separation of the projections onto the respective w vectors:



Part II: Inter-artist, intra-genre classification

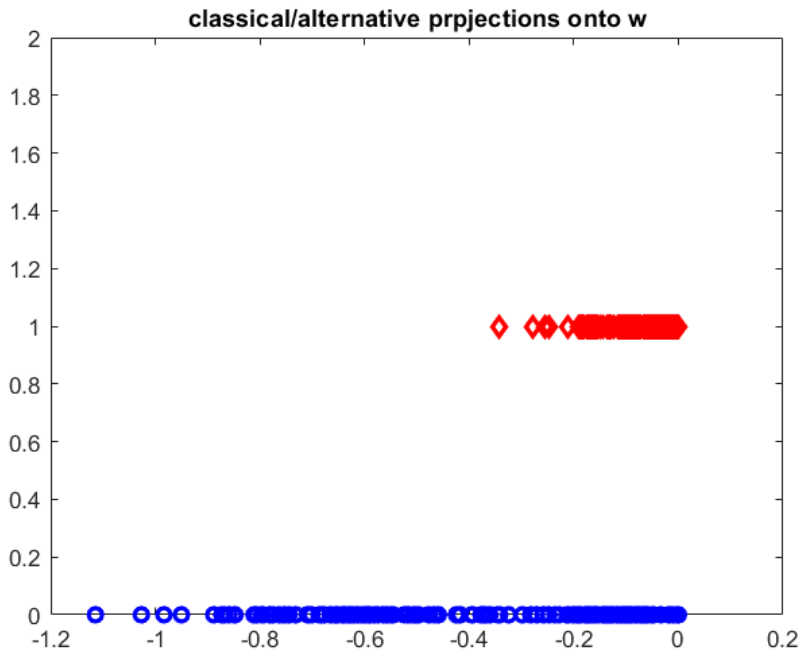
Within the genre of classical, I selected works from Beethoven, Strauss, and Vivaldi. The success rate of classification is 73%

Part II: Inter-genre classification

I aimed to select somewhat different genres: 80's classic alternative, classical, and Christmas music.

Average success rate: 66%

Surprisingly, the most common error was classic alternative being mistaken for classical constituting about 80% of all mistakes. This makes sense considering the classical/alternative LDA projection consisting of entirely overlap:



Summary and conclusion

This project provided further demonstration that principal component analysis is an incredibly effective means of analyzing sets of data and approximating underlying trends within. In conjunction with the signal analysis tools such as the windowed fft, we're able to lever some more intuitive aspects of music, namely time and frequency for the sake of classification. This project also was a first example of classification by means of linear discriminant analysis: a powerful intersection of optimization and statistics that should prove useful in the future.

References:

Dean Martin – Making Spirits Bright
Mitch Miller and the Gang – Holiday Sing Along with Mitch
Harry Connick, Jr – When my Heart Finds Christmas
Masters of Classical Music Box Set
vol. 3 Beethoven
vol. 4 Johann Strauss
vol. 5 Richard Wagner
vol. 6 Tchaikovsky
vol. 7 Vivaldi
ABBA – GOLD: Greatest Hits
The Beatles – 1967 – 1970
Classic Alternatives 80's Retro Hits

Appendix A. MATLAB Functions

`pspectrum(clip,Fs,'spectrogram') - a convenient way of creating spectrograms`

imresize(Sp,[n_freq n_times]) – an efficient and practical way of down sampling spectrograms
randsample(n,k) – retrns index values for a uniform sample of a data set. Useful when working with data sets larger than needed.

Appendix B. MATLAB Code and Data

```
%% hw4setup.m
%%
clip_num = 1;
n_freq = 100;
n_times = 100;
DataOut = spalloc(n_freq*n_times,200,4000);
for j = 1:10
    [S,Fs] = audioread(strcat('Christmas\C',int2str(j),'.mp3'));
    S = mean(S,2);
    n = length(S);
    ntrim = n-mod(n,5*Fs);
    S = S(1:ntrim);
    %V = reshape(V,[ntrim/(5*Fs) Fs*5]);
    n_clips = ntrim/(5*Fs);
    smpl = randsample(n_clips,10);
    for k = 1:10%n_clips
        k = smpl(k);
        clip = S(5*(k-1)*Fs+1:5*k*Fs);
        Sp = pspectrum(clip,Fs,'spectrogram');
        %do away with all all week signals for sparsity
        Sp = Sp.*(log(Sp)>-9);
        %Throw away high frequency information
        Sp = Sp(1:100,:);
        %downsample image
        Sp = imresize(Sp,[n_freq n_times]);
%         imagesc(Sp)
%         nnz(Sp)
%         pause(1)
        %turn image into row vector for SVD
        Sp = sparse(reshape(Sp,n_freq*n_times,1));
        %keep only clips that are representative of music
        %checks the norm to discard clips where not much is happening
        %if norm(Sp) > 0.1
            DataOut(:,clip_num) = Sp;
            clip_num = clip_num+1;
        %end
        [j k norm(Sp) clip_num]
    end
    %keep only about 2% of the data
```

```
    %Get rid of overtones and decrease resolution
End

%% hw4classification.m
%% Setup clips
%  Seperate training data from test data
load('ABBAClips.mat');
load('BeethovenClips.mat');
load('BeatlesClips.mat');
AClips = full(ABBAClips);
BClips = full(BClips);
CClips = full(BeatlesClips);
trainAind = randsample(100,80);
trainBind = randsample(100,80);
trainCind = randsample(100,80);
trainA = AClips(trainAind);
trainB = BClips(trainBind);
trainC = CClips(trainCind);
testA = AClips;
testA(:,trainAind) = [];
testB = BClips;
testB(:,trainBind) = [];
testC = CClips;
testC(:,trainCind) = [];
TestSet = [testA testB testC];
%beethoven = 1, vivaldi = 2, strauss = 3;
hiddenlabels = [ones(1,20) 2*ones(1,20) 3*ones(1,20)];
rind = randperm(60);
TestSet = TestSet(:,rind);
hiddenlabels = hiddenlabels(rind);
%% Run PCA
feature = 20; % number of PCA modes
[Uab,Sab,Vab,tab,wab,sortAb,sortaB] =
trainer(full(AClips),full(BClips),feature);
[Uac,Sac,Vac,tac,wac,sortAc,sortaC] =
trainer(full(AClips),full(CClips),feature);
[Ubc,Sbc,Vbc,tbc,wbc,sortBc,sortbC] =
trainer(full(BClips),full(CClips),feature);
%% Plot first four principal components
for k = 1:4
    subplot(2,2,k)
    ut1 = reshape(Uac(:,k),100,100);
    ut2 = rescale(ut1);
    imshow(ut2)
end
```

```
%% Plot singular values
figure(2)
subplot(2,1,1)
plot(diag(Sab),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 100])
subplot(2,1,2)
semilogy(diag(Sab),'ko','Linewidth',2)
set(gca,'Fontsize',16,'Xlim',[0 100])
title('Singular values of')
%% Plot projections of beethoven and vivaldi onto first 3 modes
figure(3)
for k=1:3
    subplot(3,2,2*k-1)
    plot(1:50,Vab(1:50,k),'ko-')
    subplot(3,2,2*k)
    plot(101:150,Vab(101:150,k),'ko-')
end
subplot(3,2,1), set(gca,'Ylim',[-.15 0],'Fontsize',12),
title('beethoven')
subplot(3,2,2), set(gca,'Ylim',[-.15 0],'Fontsize',12),
title('vivaldi')
subplot(3,2,3), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
subplot(3,2,4), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
subplot(3,2,5), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
subplot(3,2,6), set(gca,'Ylim',[-.2 .2],'Fontsize',12)
%% Plot dog/cat projections onto w
figure(4)
subplot(3,1,1)
plot(sortAb,zeros(100),'ob','Linewidth',2)
hold on
plot(sortaB,ones(100),'dr','Linewidth',2)
ylim([0 2])
ylabel('abba-beethoven')
title('3 group projections onto w - band')
subplot(3,1,2)
plot(sortAc,zeros(100),'ob','Linewidth',2)
hold on
plot(sortaC,ones(100),'dr','Linewidth',2)
ylim([0 2])
ylabel('abba-beatles')
subplot(3,1,3)
plot(sortBc,zeros(100),'ob','Linewidth',2)
hold on
plot(sortbC,ones(100),'dr','Linewidth',2)
ylim([0 2])
```

```
ylabel('beethoven-beatles')
%% Test classifier
figure(1)
for k = 1:9
    subplot(3,3,k)
    test=reshape(TestSet(:,k+5),100,100);
    imshow(test)
end

TestNum=size(TestSet,2);
%Test_wave = dc_wavelet(TestSet); % wavelet transformation
%TestMat = U'*TestSet;  % PCA projection
%pval = w'*TestMat;  % LDA projection

% Beethoven = 0, Vivaldi = 1
%ResVec = (pval>threshold)
vecs = classify3(TestSet,Uab,Uac,Ubc,tab,tac,tbc,wab,wac,wbc);
[~,result] = max(vecs,[],1);

disp('Number of mistakes')
errNum = sum(abs(result-hiddenlabels)>0)

disp('Rate of success');
sucRate = 1-errNum/TestNum
%%
function vec = classify3(TestSet,Uab,Uac,Ubc,tab,tac,tbc,wab,wac,wbc)
    TMab = Uab'*TestSet;
    TMac = Uac'*TestSet;
    TMbc = Ubc'*TestSet;
    PVab = wab'*TMab;
    PVac = wac'*TMac;
    PVbc = wbc'*TMbc;
    %larger values of PV-t favor second group
    % eg. if PV > t, then classify as grp 2
    a = -(PVab-tab) - (PVac-tac);
    b = PVab-tab - (PVbc-tbc);
    c = PVbc-tbc + PVac-tac;
    vec = [a;b;c];
end
function [U,S,V,threshold,w,sortA,sortB] = trainer(A0,B0,feature)
    nd = size(A0,2); nc = size(B0,2);

    [U,S,V] = svd([A0 B0],'econ');

    sounds = S*V'; % projection onto principal components
```

```
    U = U(:,1:feature);
    As = sounds(1:feature,1:nd);
    Bs = sounds(1:feature,nd+1:nd+nc);

    md = mean(As,2);
    mc = mean(Bs,2);

    Sw = 0; % within class variances
    for k=1:nd
        Sw = Sw + (As(:,k)-md)*(As(:,k)-md)';
    end
    for k=1:nc
        Sw = Sw + (Bs(:,k)-mc)*(Bs(:,k)-mc)';
    end

    Sb = (md-mc)*(md-mc)'; % between class

    [V2,D] = eig(Sb,Sw); % linear discriminant analysis
    [~,ind] = max(abs(diag(D)));
    w = V2(:,ind); w = w/norm(w,2);

    vA = w'*As;
    vB = w'*Bs;

    if mean(vA)>mean(vB)
        w = -w;
        vA = -vA;
        vB = -vB;
    end
    % A < threshold < B

    sortA = sort(vA);
    sortB = sort(vB);

    t1 = length(sortA);
    t2 = 1;
    while sortA(t1)>sortB(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold = (sortA(t1)+sortB(t2))/2;
end
```