Principal Component Analysis as a Classifier for Music

Ofek Inbar

Abstract

We explored the Principal Component Analysis of various songs by various artists in attempt to build a classifier for music.

1 Introduction and Overview

After selecting a number of songs by various artists to use for training data, we compute spectrograms for them. After doing this, we break these spectrograms down into their Principal Components in order to compare them to each other. Finally, we compare a number of songs not present in the training set to see if we can determine which artist produced it.

2 Theoretical Background

Principal Component Analysis is a method that enables us to take a multi-dimensional signal and break it down into a set of orthogonal “modes” and corresponding scalar values (the strength of the mode in the signal) onto which we can project the original data to get an idea for the internal “axes” it abides by.

The first step (after normalizing the data) is to apply Singular Value Decomposition to our signal data. This produces a square matrix U corresponding to the various “modes” found in the signal, a digonal matrix S whose diagonal

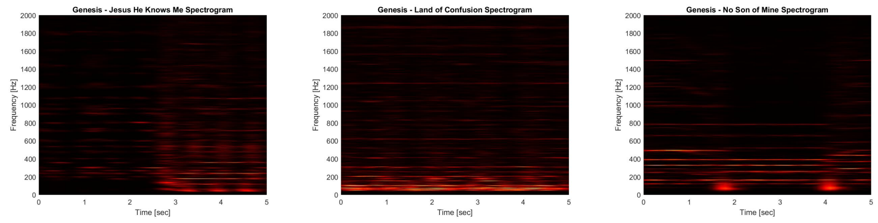

Figure 1: Spectrograms for Genesis’ “Jesus He Knows Me”, “Land of Confusion”, and “No Son of Mine” (left to right)

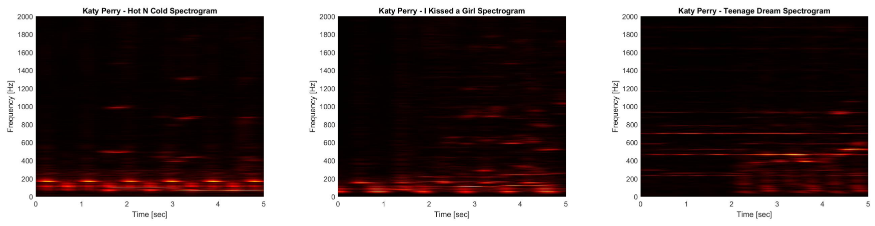

Figure 2: Spectrograms for Katy Perry's "Hot N Cold", "I Kissed a Girl", and "Teenage Dream" (left to right)

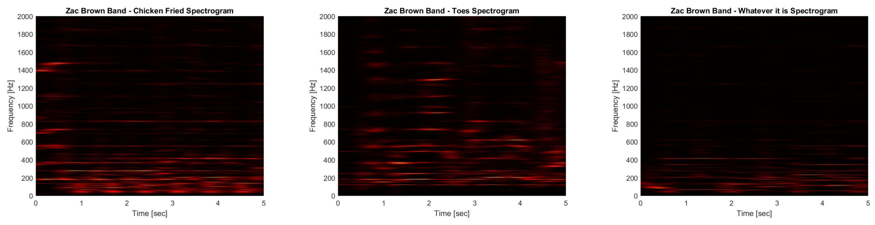

Figure 3: Spectrograms for Zac Brown Band's "Chicken Fried", "Toes", and "Whatever it is" (left to right)

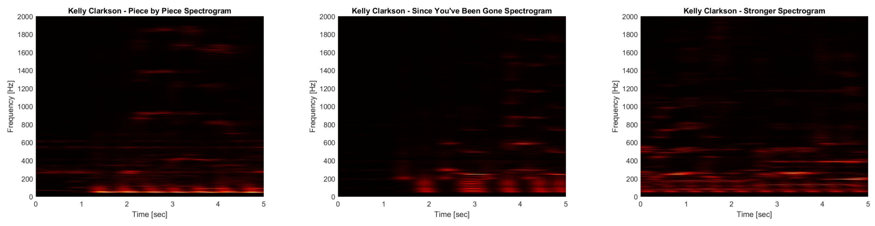

Figure 4: Spectrograms for Kelly Clarkson's "Piece by Piece", "Since U Been Gone", and "Stronger" (left to right)

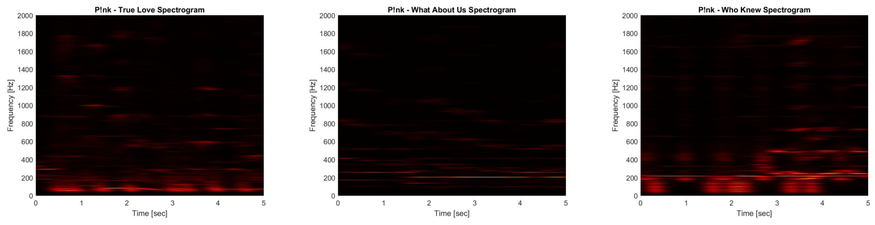

Figure 5: Spectrograms for Katy Perry's "Hot N Cold", "I Kissed a Girl", and "Teenage Dream" (left to right)

elements are scalars for the respective strength of the "modes" in U, and a matrix V whose dimensions match the transposed dimensions of the original signal data.

3 Algorithm Implementation and Development

The first step was to read in the audio data and generate a spectrogram. Then, we took the spectrograms of all 9 songs (each part had 3 artists with 3 songs each) and combined it into one big matrix. After taking the SVD of this matrix, we are able to know the principal components of the music.

This being done, we are then able to determine the "influence" of each "feature" of our matrix on each principal component, by measuring the magnitude of the projection of that feature onto the principal component. We then multiply the feature expression for each song by that feature's "influence" on each principal component, and then sum them up we can get a vector corresponding to that song's representation with respect to each principal components.

The algorithm we used to classify songs was to measure the distance between its vector representation relative to the principal components and that of each other song. The song with the least distance was chosen and the classifier output the artist that produced it.

4 Computational Results

For Part I of this assignment we were tasked with classifying a song given training data of three artists from three different genres. We selected Genesis (classic rock), Katy Perry (mid-2000s pop), and Zac Brown Band (country). The spectrograms for the songs we selected are in Figures 1, 2, and 3, respectively.

The test data that we attempted to classify in the section were "I Can't Dance" by Genesis, "Firework" by Katy Perry, and "Day that I Die" by Zac Brown Band. The following table shows the classifications:

Song	I Can't Dance	Firework	Day that I Die
Artist	Genesis	Katy Perry	Zac Brown Band
Classified Artist	Katy Perry	Katy Perry	Genesis
Correct?	No	Yes	No

Thus it turned out that our classification algorithm was woefully inadequate for this particular problem, with an accuracy of only $\frac{1}{3} = 33\%$, which is no better than randomly guessing.

Part 2 tasked us with classifying music by artists in the same genre. We selected mid-2000s pop, and so our artists were Katy Perry, Kelly Clarkson, and P!nk (see the corresponding spectrograms in Figures 2, 4, and 5). The following table shows our results for this section:

Song	Firework	My Life Would Suck Without You	Just Like Fire
Artist	Katy Perry	Kelly Clarkson	P!nk
Classified Artist	Katy Perry	Katy Perry	Katy Perry
Correct?	Yes	No	No

Once again, we only achieved an accuracy of $\frac{1}{3} = 33\%$.

5 Summary and Conclusions

Principal Component Analysis is a useful way to distill a lot of information into a few key components. As a classifier, however, we were not able to successfully determine the artist that produced our test data, with our accuracy being $\frac{1}{3}$, the same random chance.

6 Appendix A

We used `resample` to make sure all of our song data was sampled at the same rate, which made them easier to work with. We also used `squeeze` to get rid of extraneous size-1 dimensions and `svd` to break down our matrix into Singular Values. Finally, we used `sign` to get the direction of a vector and `vecnorm` to get the column-wise norms of a matrix of vectors.

7 Appendix B

The `MATLAB` code for classifying the data and producing the figures is below:

```
%% Part 1
clear variables; close all; clc;

training_data = ["genesis", "land_of_confusion", "jesus_he_knows_me", "no_son_of_mine"; ...
    "katy_perry", "hot_n_cold", "i_kissed_a_girl", "teenage_dream"; ...
    "zac_brown_band", "chicken_fried", "toes", "whatever_it_is"];
titles = ["Genesis", "Land of Confusion", "Jesus He Knows Me", "No Son of Mine"; ...
    "Katy Perry", "Hot N Cold", "I Kissed a Girl", "Teenage Dream"; ...
    "Zac Brown Band", "Chicken Fried", "Toes", "Whatever it is"];
start_times = [2, 2, 10; 35, 2, 2; 24, 40, 2];
sampling_rate = 44100/3;
audio = zeros(3, 3, 5*sampling_rate + 1);
ffts = zeros(3, 3, 5*sampling_rate + 1);

n=5*sampling_rate;
k=(1/5)*[0:(n+1)/2 -(n+1)/2:-1];
ks=fftshift(k);
```

```
for i = 1:3
    for j = 1:3
        artist = training_data(i, 1);
        song = training_data(i, j + 1);
        audio(i, j, :) = read_audio(make_audio_path(1, artist, song), ...
            start_times(i, j), ...
            sampling_rate);
        ffts(i, j, :) = abs(fft(squeeze(audio(i, j, :))'));

        tslide = 0:0.1:5;
        S = squeeze(audio(i, j, :))';
        Sgt_spec = zeros(length(tslide), n+1);

        for k=1:length(tslide)
            g = exp(-20 .* ((1:n+1)/sampling_rate - tslide(k)) .^ 2);
            Sg = g .* S;
            Sgt = fft(Sg);
            Sgt_spec(k, :) = fftshift(abs(Sgt));
        end

        f = figure(j);
        pcolor(tslide, ks, Sgt_spec(:, :).');
        shading interp;
        set(gca,'Ylim',[0 2e3]);
        colormap(hot);
        xlabel('Time [sec]');
        ylabel('Frequency [Hz]');
        title(sprintf('%s - %s Spectrogram', titles(i, 1), titles(i, 1 + j)));
        saveas(f, sprintf('%s_%s_spectrogram.jpg', artist, song));
        close(f);
    end
end

X = [squeeze(ffts(1, :, :)); squeeze(ffts(2, :, :)); squeeze(ffts(3, :, :))];
[U,S,V] = svd(X, 'econ');

influences = zeros(9, 5*sampling_rate + 1);
for i = 1:9
    projection = S(i, i) * U(:, i) * V(:, i)';
    influences(i, :) = sign(U(:, i)'*projection) .* vecnorm(projection);
end

scores = zeros(3, 3, 9);
for i = 1:3
    for j = 1:3
        scores(i, j, :) = influences*squeeze(ffts(i, j, :));
    end
end

%% Test

test_data = ["day_that_i_die", "firework", "i_cant_dance"];
test_start_times = [2, 2, 35];
test_audio = [];
test_ffts = [];
test_scores = [];

for i = 1:size(test_data, 2)
    test_audio(i, :) = read_audio("test_data/part_1/" + test_data(i) + ".mp3", ...
        test_start_times(i), ...
        sampling_rate);
    test_ffts(i, :) = abs(fft(squeeze(test_audio(i, :))'));
    test_scores(i, :) = influences*squeeze(test_ffts(i, :))';
end

for k = 1:size(test_data, 2)
    min = 1e10;
    for i = 1:3
        for j = 1:3
            squared_pieces = (squeeze(test_scores(k, :)) - squeeze(scores(i, j, :))).^2
            distance = sqrt(ones(1, 9)*squared_pieces);
            if distance < min
                min = distance;
                argmin = training_data(i, 1);
            end
        end
    end

    argmin
end

%% Part 2
clear variables; close all;

training_data = ["katy_perry", "hot_n_cold", "i_kissed_a_girl", "teenage_dream"; ...
    "kelly_clarkson", "piece_by_piece", "since_youve_been_gone", "stronger"; ...
    "pink", "true_love", "what_about_us", "who_knew"];
titles = ["Katy Perry", "Hot N Cold", "I Kissed a Girl", "Teenage Dream"; ...
    "Kelly Clarkson", "Piece by Piece", "Since You've Been Gone", "Stronger"; ...
    "P!nk", "True Love", "What About Us", "Who Knew"];
start_times = [35, 2, 2; 10, 2, 10; 10, 10, 10];
sampling_rate = 44100/3;
```

```
audio = zeros(3, 3, 5*sampling_rate + 1);
ffts = zeros(3, 3, 5*sampling_rate + 1);

n=5*sampling_rate;
k=(1/5)*[0:(n+1)/2 -(n+1)/2:-1];
ks=fftshift(k);

for i = 1:3
    for j = 1:3
        artist = training_data(i, 1);
        song = training_data(i, j + 1);
        audio(i, j, :) = read_audio(make_audio_path(2, artist, song), ...
            start_times(i, j), ...
            sampling_rate);
        ffts(i, j, :) = abs(fft(squeeze(audio(i, j, :))'));

        tslide = 0:0.1:5;
        S = squeeze(audio(i, j, :))';
        Sgt_spec = zeros(length(tslide), n+1);

        for k=1:length(tslide)
            g = exp(-20 .* ((1:n+1)/sampling_rate - tslide(k)) .^ 2);
            Sg = g .* S;
            Sgt = fft(Sg);
            Sgt_spec(k, :) = fftshift(abs(Sgt));
        end

        f = figure(j);
        pcolor(tslide, ks, Sgt_spec(:, :).');
        shading interp;
        set(gca,'Ylim',[0 2e3]);
        colormap(hot);
        xlabel('Time [sec]');
        ylabel('Frequency [Hz]');
        title(sprintf('%s - %s Spectrogram', titles(i, 1), titles(i, 1 + j)));
        saveas(f, sprintf('%s_%s_spectrogram.jpg', artist, song));
        close(f);
    end
end

X = [squeeze(ffts(1, :, :)); squeeze(ffts(2, :, :)); squeeze(ffts(3, :, :))];
[U,S,V] = svd(X, 'econ');

influences = zeros(9, 5*sampling_rate + 1);
for i = 1:9
    projection = S(i, i) * U(:, i) * V(:, i)';
    influences(i, :) = sign(U(:, i)'*projection) .* vecnorm(projection);
end

scores = zeros(3, 3, 9);
for i = 1:3
    for j = 1:3
        scores(i, j, :) = influences*squeeze(ffts(i, j, :));
    end
end

%% Test

test_data = ["firework", "just_like_fire", "my_life_would_suck_without_you"];
test_start_times = [2, 14, 2];
test_audio = [];
test_ffts = [];
test_scores = [];

for i = 1:size(test_data, 2)
    test_audio(i, :) = read_audio("test_data/part_2/" + test_data(i) + ".mp3", ...
        test_start_times(i), ...
        sampling_rate);
    test_ffts(i, :) = abs(fft(squeeze(test_audio(i, :))'));
    test_scores(i, :) = influences*squeeze(test_ffts(i, :))';
end

for k = 1:size(test_data, 2)
    min = 1e10;
    for i = 1:3
        for j = 1:3
            squared_pieces = (squeeze(test_scores(k, :)) - squeeze(scores(i, j, :))).^2
            distance = sqrt(ones(1, 9)*squared_pieces);
            if distance < min
                min = distance;
                argmin = training_data(i,  1);
            end
        end
    end

    argmin
end
```

The code above relies on the following two `MATLAB` functions that we implemented for this assignment:

```
function p = make_audio_path(part, artist, song)
    p = "training_data/part_" + part + "/" + artist + "/" + song + ".mp3";
```

```
end

function [y, Fs] = read_audio(path, secondsOffset, sampling_rate)
    [y, Fs] = audioread(path);
    y = resample(y, sampling_rate, Fs);
    Fs = sampling_rate;
    length = 5 * Fs;
    start = secondsOffset * Fs;
    y = y(start:start+length);
end
```