Project Proposal - Music Genre Classification

Krishna Chaitanya Sripada, Siva Palakurthi, Soumyatha Gavvala

November 9, 2015

1 Introduction

The rapid growth of the digital media in the past decade has made it easier for man to associate with his interests at a faster and effortless ways. Man has been finding new ways to make music, find, store it and organize it. Finding music was difficult without the name or lyrical content and this lead to the necessity of searching music by its content. Music Information Retrieval (MIR), which is extraction of information about the music from the raw data is a small but fast growing field of research in the similar application. MIR deals with analyzing, creating and categorizing music.

Our project's aim is to classify a vast collection of musical tracks by their content into different genres. We have taken 729 tracks of known labels as the training data on the basis of which we build the algorithm to categorize unknown test music files. These files were acquired from the website of the 5th International Conference on Music Information Retrieval. These 729 songs belong to six genres.

- Classical: 320 songs
- Electronics: 115 songs
- Jazz/blues: 26 songs
- Metal/punk: 45 songs
- Rock/pop: 101 songs
- World: 122 songs

For the classification of the raw training data given to us, we are using MATLAB as the platform for our algorithms. We are provided with a Music Analysis (MA) toolbox to process the songs which are Pulse Code Modulated and sample them into data that is easier to handle on MATLAB.

The issue we face is the large size of the data and the classification into genres based on the content which requires the analysis of features of the songs like bandwidth and spectral roll-off. When the songs are converted into matrices, we get a data set of high dimensions and so in order to make it easier to work on the data, we reduce the dimensionality of the data. Then we classify them into groups according to their labels based on the distances between the songs. The distance between two songs of the same genre would be less than that of two

songs of different genres. On this basis we cluster the training set into groups thus building an algorithm to classify unknown music files.

2 Distance in song-space

For the categorization of these 729 songs into their respective labels, we need an appropriate basis to differentiate them. This we do by measuring the distance between the songs. The distance between two songs of the same genre would be less than that of two songs of different genres. The distance in question is the L2 or Euclidean distance, which is the difference in the sum of squared terms between two given vectors.

One of the method we have implemented is the calculation of Frobenius norm of the L2 norm. Frobenius norm is also called as Euclidean norm but is not L2 norm. The Frobenius norm of different songs was compared to measure the distance between them.

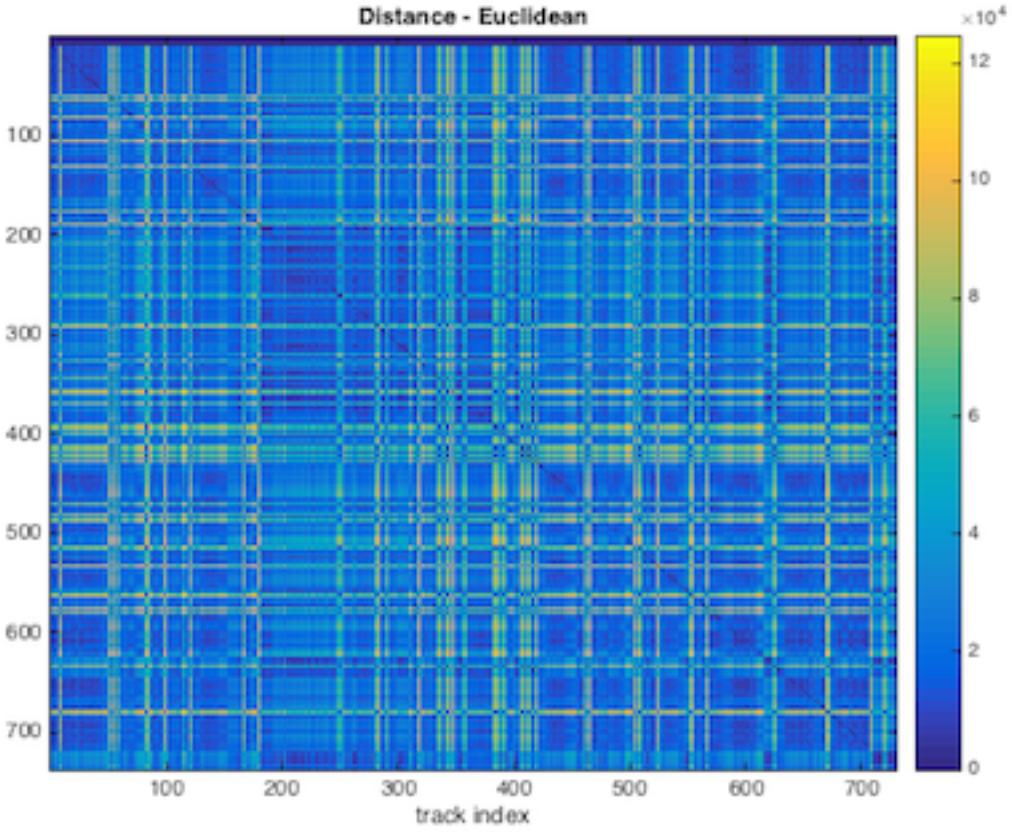

Figure 1: Graphical representation of the distance matrices we used. Dark blue regions represent songs which are close to one another while the yellow regions represent a larger distance between songs.

3 Dimension reduction

For the analysis of the training data we need to sample the given songs. We get hundreds of thousands of samples and the dimension of a song in its raw form is huge and impractical to work with directly. So we used several dimensionality reduction techniques in order to reduce the data to dimensions that allowed us to work with the hundreds of songs in the training data set.

3.1 MFCC

Mel Frequency Cepstral Coefficients (MFCCs), are a pre-processing technique that is frequently used in speech recognition and has been adapted for music information retrieval. The raw data when sampled has dimensions in terms of thousands and it is a strenuous procedure to analyze data of such high dimensions. Due to the high number of samples, the data in each sample is not sufficient for analysis. Hence we requires some pre-processing of the data in order to extract useful information.

From the MA toolbox provided to us, we make use of the MFCC function to obtain MFC coefficients. These coefficients extract information from songs that is similar to how the human ear hears music that is the content. We first transform the songs into their MFCC coefficients, effectively reducing the dimension of each song while retaining useful information.

3.2 Principal Component Analysis

Principal Component Analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

PCA [1] also called the K-L method searches for k n-dimensional orthogonal vectors that can best be used to represent the data where k $\leq$n. The original data are thus projected onto a much smaller space resulting in dimensionality reduction. For our experiments, the results of MFCC are passed through the PCA technique which even further reduced the dimension of the songs. The implementation is provided at the end of this document.

4 Statistical learning

Once the data has been sufficiently reduced in dimension, we used the K-Nearest Neighbors statistical learning method to classify the songs according to their genre. The results of the experiments are provided below.

4.1 K-Nearest Neighbors

The kNN algorithm [2] simply computes the k nearest neighbors to the song which is to be classified, where k $\in$ N is a natural number. The song is then classified into a genre based on the genre of its nearest neighbors. To avoid ties, k is usually chosen to be an odd number.

This is one of the simplest algorithms and fastest available and can be easily applied to any of our distance matrices by simply sorting the matrix to find the nearest neighbors.

5 Initial Results (extra credit)

We ran the k-Nearest Neighbors implementation over the entire dataset. The following are the results of using 90% of the data as the training set and the remaining 10% of the data as the testing set.

k	Average Accuracy
1	63.47%
3	60.84%
5	66.50%
7	62.40%
9	63.02%
11	64.19%
13	63.23%
15	63.80%

We algorithm was run on a average of 10 times for every odd value of k from the range of 1 to 15.

6 Discussion

From the initial experiments, we see that the accuracy using the kNN algorithm falls in the range of 60-66% for different values of k. We understand that this accuracy is attributed to the fact that we used a part of the training data for testing and once the testing data is out, we expect the accuracies to improve. But we believe using our current testing data, the accuracy can be improved and would thus like to explore other dimensionality reduction techniques like Latent Dirichlet Allocation(LDA), spectral techniques such as LLE and Laplacian Eigen Maps. Random projections is also something we would look into. The idea is to find out which technique would best describe the data and yet not lose any valuable information about the song. Also we would like to explore other distance measures such as a mixture of multivariate Gaussian distributions to each song along with K-L divergence.

Apart from this, we believe providing more features would help the classifier. We would try out some of the features like bandwidth of the song, spectral roll-off, spectral flux and loudness. Also other classifiers like Neural Networks, Random Forest based classification and Spectral Clustering would to looked into to see if we can improve the accuracy of our classification by using either of them.

References

[1] `http://lvdmaaten.github.io/drtoolbox/`

[2] `http://www.mathworks.com/matlabcentral/fileexchange/45831-matlab-audio-analysis-library`

7 Source Code

1. Distance in Song Space

```
load('/Users/krishnachaitanyasripada/GitHub/
   highdimensionaldataset/pcaResult/pcaResults.mat');
variables = who;
distance = [];
for i=1:length(variables)
   if strncmpi(variables{i}, 'mfcc_', 5)==1
       matrix1 = eval(char(variables{i}));
       values = [];
       for j=1: length(variables)
            if strncmpi(variables{j}, 'mfcc_', 5)==1
                matrix2 = eval(char(variables{j}));
                euclideanDistance = sqrt(sum((matrix1 -
                   matrix2) .^ 2));
                frobeniusNorm = norm(euclideanDistance, '
                   fro');
                values = [values frobeniusNorm];
            end
       end
       distance(i,:) = values;
   end
end
imagesc(distance);
colorbar;
```

2. MFCC and PCA calculation

```
files = dir('./tracks/*.wav');
filename = cell(1, length(files));
%save('mfccResults.mat');
save('./pcaResult/pcaResults.mat');
for k = 1:length(files)
    file = strcat('./tracks/',files(k).name);
    [~, name, ~] = fileparts(file);
    [mfcc_result,~] = mfcc_coeffs(file);
    pcaData = DimensionReduction(mfcc_result, 'pca', 79);
    varname = ['mfcc_' name];
    assignin('base', varname, pcaData);
    save('./pcaResult/pcaResults.mat', varname, '-append');
    clear mfcc_result file name varname pcaData
end

function [matrixDR] = DimensionReduction(matrix, method,
   num_dim)
if (strcmp(method, 'pca'))
    matrixDR = pca(matrix, num_dim);
```

```
end
end
```

3. kNN classification

```
final_results = cell(15,2);
for K_n=1:15
    all_acc = 0;
    for loop=1:5
        % Training Data
        f = {};
        f.classNames = {'classical','electronic','
           jazz_blues','metal_punk','rock_pop','world'};
        f.features = {[],[],[],[],[],[]};
        load('/Users/krishnachaitanyasripada/GitHub/
           highdimensionaldataset/pcaResult/pcaResults.mat
           ');
        load_ground_truth;
        variables = who;
        variables_90_percent = datasample(variables,656,'
           Replace', false);
        variables_10_percent = setdiff(variables,
           variables_90_percent);
        for i=1:length(variables_90_percent)
            if strncmpi(variables_90_percent(i), 'mfcc_',
               5)==1

               % find song type, find index in song, and
                  get corresponding index value
               % from type
               temp = strrep(variables_90_percent(i), '
                  mfcc_', '');
               filename = strcat('tracks/', temp);
               filename = strcat(filename, '.mp3');

               songid = findPosition(song, filename);
               typeid = findPosition(f.classNames, type(
                  songid));

               typeMat = cell2mat(f.features(typeid));
               [rows, cols] = size(typeMat);

               varialbeMatrix = eval(char(
                  variables_90_percent(i)));
               typeMat(:, cols+1) = varialbeMatrix(:,1);
               f.features(typeid) = {typeMat};
            end
        end
```

```
clearvars -except f variables_10_percent K_n
   all_acc loop final_results
% Testing Data
%load('/Users/krishnachaitanyasripada/GitHub/
   highdimensionaldataset/testdata/pcaResult/
   pcaResults_35.mat');
%variables = who;
testsample = [];
verification = cell(length(variables_10_percent),2)
   ;
for i=1:length(variables_10_percent)
    if strncmpi(variables_10_percent{i}, 'mfcc_',
       5)==1
       temp = load('/Users/krishnachaitanyasripada/
          GitHub/highdimensionaldataset/pcaResult/
          pcaResults.mat', variables_10_percent{i})
          ;
       varialbeMatrix = getfield(temp,
          variables_10_percent{i});
       % Classifying
       [~,winnerClass] = classify('knn', f,
          varialbeMatrix(:,1), K_n);
       temp = strrep(variables_10_percent(i), '
          mfcc_', '');
       filename = strcat('tracks/', temp);
       filename = strcat(filename, '.mp3');
       verification{i, 1} = filename;
       verification{i, 2} = winnerClass;
    end
end
testresults = verification.';
result = reshape(testresults(~cellfun(@isempty,
   testresults)),2, [])';

%Accuracy
counter = 0;
load_ground_truth;
for k=1: length(result)
    rowContent = result(k,:);
    songid = findPosition(song, rowContent{1});
    typeid = findPosition(f.classNames, type(songid
       ));
    if(typeid==rowContent{2})
        counter = counter + 1;
    end
end
```

```
        accuracy = counter/length(result);
        all_acc = all_acc + accuracy;
    end
    average_acc = all_acc/5;
    final_results(K_n,1) = {K_n};
    final_results(K_n,2) = {average_acc};
end

function [probabilites, winnerClass] = classify(method, f,
   testsample, k)

if strcmp(method,'knn')
    [probabilites, winnerClass]=classifyKNN_D_Multi(f.
       features, testsample, k, 0);
end

end

function pos = findPosition(data, str)

strPos = ismember(data, str);

for pos=1:length(strPos)
   if strPos(pos) == 1
       break
   end
end
end
```