Introduction To Machine Learning Class Classification

Casper Hsiao

2020-03-06

Abstract: The objective of this project is to implement linear discriminant analysis (LDA) to classify music by artists and genres. The implementation incorporates the analysis methods from previous projects such as spectrogram, singular value decomposition (SVD) and principal component analysis (PCA). The music pieces were fragmented into clips and the spectrogram of each clip was obtained. Furthermore, SVD and PCA was performed on the spectrograms to represent the sample with their principal components that is used for the LDA. The results of the implementation were able to classify music genre up to an accuracy of 80%, and classify artists with an accuracy of 45%.

I. Introduction and Overview

Three different tests were performed to verify the implementation of the LDA. The first test was to classify the music pieces by artist of different genres. The second test was to classify the music by artist of the same genre. The last test was to classify the music by genre. In each test, the training music pieces were fragmented into 5 second clips for sampling and the spectrogram of each clip was obtained to perform SVD and PCA on each sample. Furthermore, the featured projection of principal components of each sample were utilized as the classification weight which represents each sample for the LDA to generate thresholds. The new test music pieces were also fragmented into sample to perform SVD and PCA with the same method. Using LDA, the projection of the test sample were compared to the thresholds obtained to classify the sample.

II. Theoretical Background

The PCA is based on the theories of linear algebra and matrix decomposition, therefore, the SVD is introduced complementary.

II.A. Singular Value Decomposition (SVD)

The singular value decomposition is a factorization of a matrix into a number of constitutive components. Furthermore, it is a transformation that scales and rotates a given set of vectors. The full SVD is shown in Eq. 1 where $U \in \mathbb{C}^{m\times m}$ is unitary, $V \in \mathbb{C}^{n\times n}$ is unitary and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal. A unitary matrix means that its conjugate transpose is equal to its inverse. A diagonal matrix means that its off-diagonal entries are zeros.

$$A = U\Sigma V^* \tag{1}$$

The SVD of matrix A is performed through applying V^* for unitary transformation to preserve the unit sphere, and a scaling transformation to create an ellipse with principal semiaxes determined by Σ. Lastly, a unitary rotational transformation is applied to hyperellipse by U. Furthermore, the theorem states that every matrix $A \in \mathbb{C}^{m\times n}$ has a SVD.

II.B. principal Component Analysis (PCA)

The PCA reduces a complex high-dimensional data to a low-dimensional data that consists the dynamics and behaviors of the system. Therefore, the PCA is utilized to analyze unknown and potentially low-dimensional data.

The covariance matrix determines the similarity between two variables. The covariance matrix is calculated as shown in Eq.2. $X \in \mathbb{R}^{m\times n}$ is the vector storing the sample points of the variables where m represents the number of measurement types and n is the number of sample points.

$$C_X = \frac{1}{n-1} XX^T \tag{2}$$

The covariance matrix C_X is a square, symmetric $m \times m$ matrix where the diagonal entries represents the variances of associated measurement type, and the off-diagonal entries are the covariances between different measurement types. A large off-diagonal entry represents redundancy in the measurement types, and small off-diagonal entry represents Independence between the measurement types. Furthermore, large diagonal entry represents strong fluctuations in that variable, which would be the dynamics of interest. As a result, SVD is utilized to diagonalize the covariance matrix to obtain the variances of the variables.

II.C. Linear Discriminant Analysis (LDA)

The LDA enables classification of different labeled data. The objective of linear discriminant analysis is to separate the projection of the data of different classes. Two key concepts of the LDA are to find the projection that maximizes the distances between the inter-class data, and that minimizes the distances between intra-class data. Eq. 3 showed the equation of the projection $\boldsymbol{w}$ with respect to the inter-class data $\boldsymbol{S_B}$ and intra-class data $\boldsymbol{S_w}$ matrices.

$$\boldsymbol{S_B w} = \lambda \boldsymbol{S_w w} \tag{3}$$

Eq. 4 showed the equation of $\boldsymbol{S_B}$ and $\boldsymbol{S_w}$ of an two-class LDA.

$$\boldsymbol{S_B} = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T \qquad \text{and} \qquad \boldsymbol{S_w} = \sum_{j=1}^{2} \sum_{\boldsymbol{x}} (\boldsymbol{x} - \mu_j)(\boldsymbol{x} - \mu_j)^T \tag{4}$$

III. Algorithm Implementation and Development

For test one, the three different artists of different genres I chose were Galantis (EDM), Beethoven (classical) and Eminem (rap). The second test, the three different artists of the rap genre I chose were Eminem, Chance and Pusha T. For the last test, the three different genres I chose were EDM, classical and rap. For each test, I obtained at least two music pieces for each class as training data, and one music piece for each class as testing data for verification. All the data were fragmented where each music pieces where sampled into 10 pieces of 5-second clip. The spectrograms of the clips were obtained using the MATLAB built-in function. SVD and PCA were performed on the clips of the training data to obtain the projected principal components of each clip. Furthermore, the featured principal component projection were utilized as the representation (weights) of the clip in the LDA. Using the concept in Eq. 4, the scatter matrices of the inter-class and intra-class data. The LDA projection matrix is obtained using the scatter matrices and Eq.3. Furthermore, the weights (featured principal components) of the clips were projected by the LDA projection matrix. The LDA projections was then utilized to generate the mean projection value of each class. The mean values were use to sort the classes in order. Lastly, the threshold value between each class were determined by the intersection of the projection of each class. As a result, the thresholds were used as the classifier between the music pieces.

To verify the classifier, the testing music clips were also processed with the same method above to obtain the LDA projections. Lastly, the LDA projection of each clips were compared to the thresholds to determine the class of the clips.

IV. Computational Results

In test one, a total of 60 training clips was sampled evenly among six different training music pieces, two music pieces from each of the three artist (Galantis, Beethoven & Eminem). The first three modes of the principal projection of the clips were plotted as shown in Fig. 1. It is apparent that the projection of the Beethoven clips are grouped together, and the Galantis and Eminem clips are scattered. Therefore, it would be relatively easier to classify Beethoven as compared to classify Galantis or Eminem.

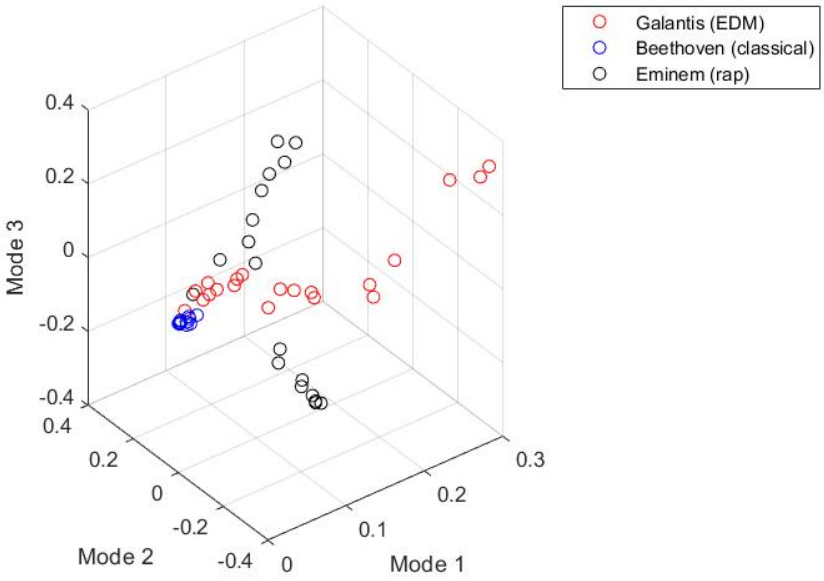


Fig. 1: The first three modes of the principal projection of the clips in test 1.

A total of 60 testing clips was sampled among three new testing music pieces of the three artists. The overall accuracy of the classification was 80%. Furthermore, the distribution of the principal projection was verified because the accuracy of classifying Eminem and Galantis is the lowest, 60% and 80%.

In test two, a total of 60 training clips was sampled evenly among six different training music pieces, two music pieces from each of the three artist (Eminem, Chance & Pusha T). The first three modes of the principal projection of the clips were plotted as shown in Fig. 2. It is apparent that the projection of the all the clips are scattered. Therefore, it would be relatively difficult to classify the clips as compared in test 1.

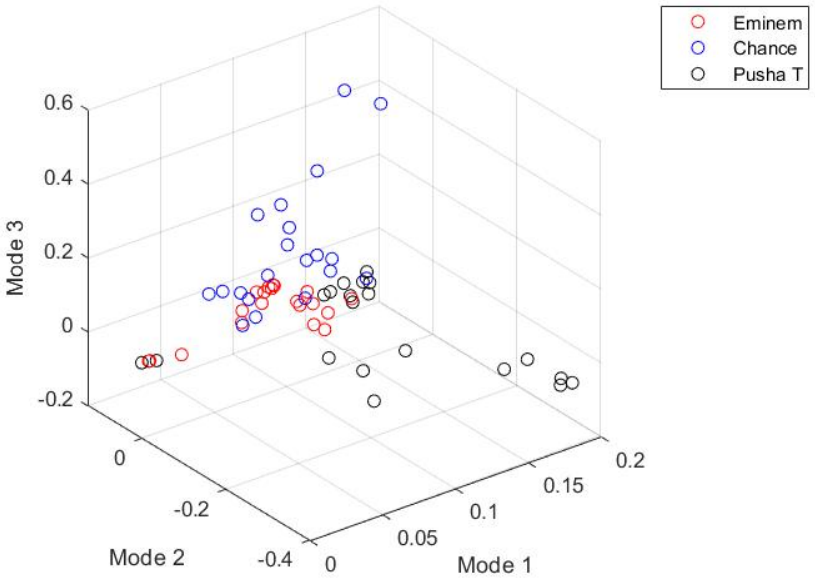


Fig. 2: The first three modes of the principal projection of the clips in test 2.

A total of 30 testing clips was sampled among three new testing music pieces of the three artists. The overall accuracy of the classification was 45%. Furthermore, the distribution of the principal projection was verified because the accuracy is much lower as compared to test 1.

In test three, a total of 90 training clips was sampled evenly among nine different training music pieces, three music pieces from each of the three genres (rap, classical & rap). The first three modes of the principal projection of the clips were plotted as shown in Fig. 3. It is apparent that the projection of the classical clips are grouped together, and the EDM and rap clips are scattered. Therefore, it would be relatively easier to classify classical music pieces as compared to classify EDM or rap.

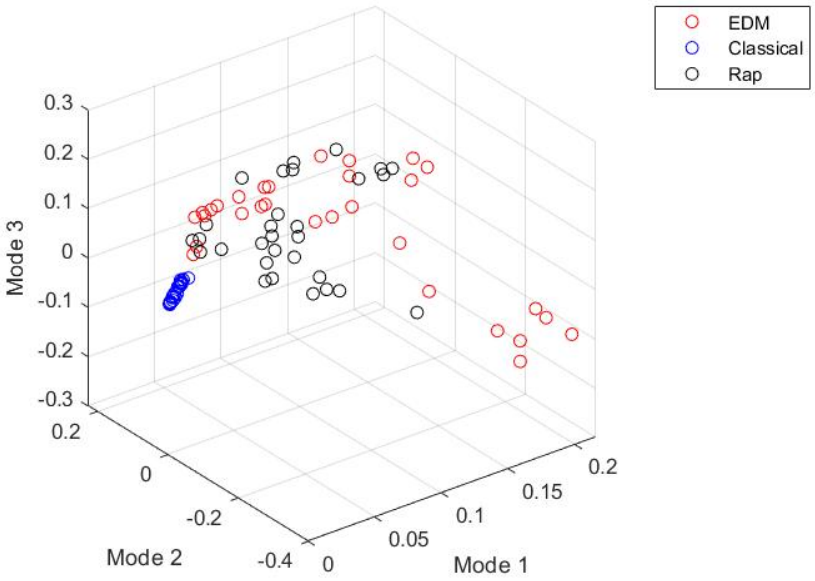


Fig. 3: The first three modes of the principal projection of the clips in test 3.

A total of 30 testing clips was sampled among three new testing music pieces of the three artists. The overall accuracy of the classification was 70%. Furthermore, the distribution of the principal projection was verified because the accuracy of classifying EDM and rap is the lowest, 30% and 80%.

V. Summary and Conclusions

The implementation of LDA on three different tests demonstrate the ability to classify music artist and genre to a certain extend. The classification accuracy were around 75% for test 1 and 3, and 45% for test 2. The classes of test 2 were close to each other as the clips were from the same genre, which resulted a relatively low accuracy of classification. On the other side, the classes of test 1 and test 2 were relatively distinct. As a result, the accuracy of classification is relatively high.

Appendix A

dir : loads the file directory from local machine.
$svd(x, `econ')$: perform SVD on the matrix x and return reduced U, S and .
$diag(x)$: the diagonal values of matrix X.
$sum(x)$: summation of x.
$reshape(x, sz)$: reshape matrix X to the specified size sz.
$audioread(x)$: read audio file x.
$mean(x)$: the mean of the matrix$x$.
$spectrogram(x)$: short-time Fourier transform of the matrix x.

Appendix B

Contents


```
clc; clear all; close all;
```

Load music and compute spectrogram

```
pathTrain = 'songs\test1\train\*.mp3';
nClips = 10;
clipLength = 5;
[Train, fNameTrain] = spectro(pathTrain, nClips, clipLength);
```

Perform SVD on the spectrogram

```
[U, S, V] = svd(Train, 'econ');
diagS = diag(S)/sum(diag(S));

figure(1)
plot(diagS, 'ro');
grid on;
xlabel('Modes');
ylabel('SVD value');

figure(2)
plot3(V(1:(2*nClips), 1), V(1:(2*nClips), 2), V(1:(2*nClips), 3), 'ro');
grid on;
hold on;
plot3(V((2*nClips + 1):(4*nClips), 1), V((2*nClips + 1):(4*nClips), 2), ...
    V((2*nClips + 1):(4*nClips), 3), 'bo');
plot3(V((4*nClips + 1):(6*nClips), 1), V((4*nClips + 1):(6*nClips), 2), ...
    V((4*nClips + 1):(6*nClips), 3), 'ko');
legend('Galantis (EDM)', 'Beethoven (classical)', 'Eminem (rap)')
xlabel('Mode 1');
ylabel('Mode 2');
zlabel('Mode 3');
```

Contents


```
clc; clear all; close all;
```

Load music and compute spectrogram

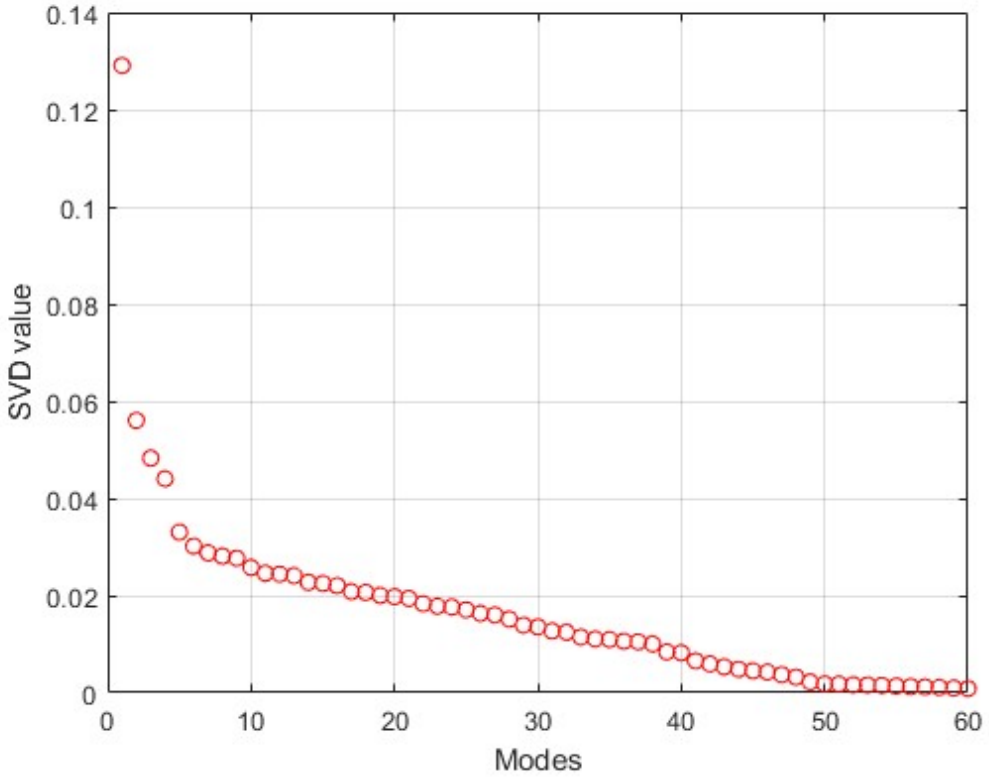
```
pathTrain = 'songs\test1\train\*.mp3';
nClips = 10;
clipLength = 5;
[Train, fNameTrain] = spectro(pathTrain, nClips, clipLength);
```

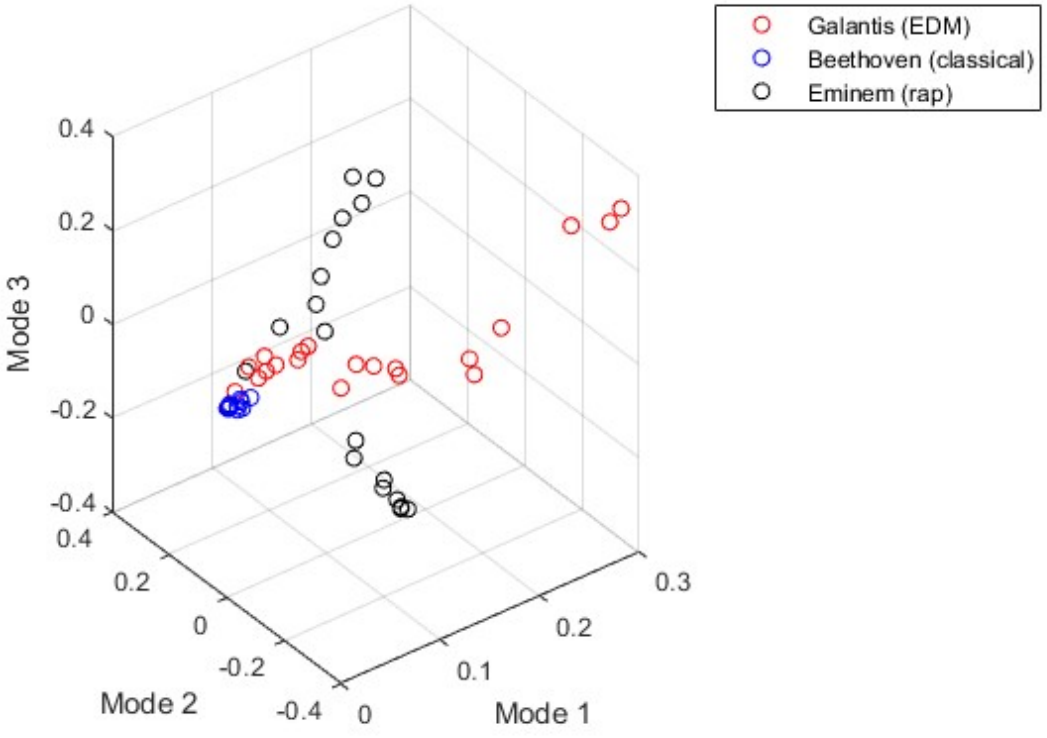
Perform SVD on the spectrogram

```
[U, S, V] = svd(Train, 'econ');
diagS = diag(S)/sum(diag(S));

figure(1)
plot(diagS, 'ro');
grid on;
xlabel('Modes');
ylabel('SVD value');

figure(2)
plot3(V(1:(2*nClips), 1), V(1:(2*nClips), 2), V(1:(2*nClips), 3), 'ro');
grid on;
hold on;
plot3(V((2*nClips + 1):(4*nClips), 1), V((2*nClips + 1):(4*nClips), 2), ...
    V((2*nClips + 1):(4*nClips), 3), 'bo');
plot3(V((4*nClips + 1):(6*nClips), 1), V((4*nClips + 1):(6*nClips), 2), ...
    V((4*nClips + 1):(6*nClips), 3), 'ko');
legend('Galantis (EDM)', 'Beethoven (classical)', 'Eminem (rap)')
xlabel('Mode 1');
ylabel('Mode 2');
zlabel('Mode 3');
```


SVD value
Modes
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
10
20
30
40
50
60


Galantis (EDM)
Beethoven (classical)
Eminem (rap)
Mode 3
Mode 2
Mode 1
0.4
0.2
0
-0.2
-0.4
0.3
0.1

Load test

```
pathTest = 'songs\test1\test\*.mp3';
[Test, fNameTest] = spectro(pathTest, 2*nClips, clipLength);
```

Build Classifier

```
feature = 3; % 3 = 0.80
[U1, S1, V1, threshold12, threshold23, w, sort1, sort2, sort3] ...
    = artist_trainer(Train, nClips, feature);
pval = w'*(U1'*Test);

resultL = length(pval);
result = strings(1, resultL);
for i = 1:length(pval)
    pval1 = pval(i);
    if pval1 >= threshold23
        result(i) = "C";
    elseif pval1 >= threshold12
        result(i) = "E";
    else
        result(i) = "P";
    end
end

answer = strings(1, resultL);
for i = 1:length(result)
    if i <= resultL/3
        answer(i) = "E";
    elseif i <= resultL*2/3
        answer(i) = "C";
    else
        answer(i) = "P";
    end
end
compare = result == answer;
disp(compare);
accuracy = length(find(compare == 1))/length(result);
disp(accuracy)



function [Data, fName] = spectro(path, nClips, clipLength)
    folder = dir(path);
    nSongs = length(folder);
    Data = [];
    fName = [];
    for i = 1:nSongs
        fname = folder(i).name;
        fName = [fName, convertCharsToStrings(fname)];
        [y, Fs] = audioread([path(1:end -5), fname]);
```

```
        y = (y(:, 1) + y(:, 2))/2;
        s = y';
        t = (1:length(s))/Fs;
        lengthSong = t(end);
        totalClips = lengthSong/clipLength;
        clipSpace = totalClips/nClips;
        tempData = [];
        for j = 1:clipSpace:totalClips
            sStart = floor((j - 1)*clipLength);
            if j == 1
                sStart = 1;
            end
            sEnd = sStart + clipLength;
            clip = s(1, sStart*Fs : sEnd*Fs);
            clipSpec = abs(spectrogram(clip));
            clipSpec = reshape(clipSpec, ...
                size(clipSpec, 1)*size(clipSpec, 2), 1);
            tempData = [tempData, clipSpec];
        end
        Data = [Data, tempData - mean(tempData(:))];
    end
end


function [U, S, V, threshold12, threshold23, w, sort1, sort2, ...
    sort3] = artist_trainer(Data, nSample, feature)

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

    n = 2*nSample;

    Artist = S*V';
    U = U(:, 1:feature);
    A1 = Artist(1:feature, 1:n);
    A2 = Artist(1:feature, (n + 1):(n + n));
    A3 = Artist(1:feature, (n + n + 1):(n + n + n));

    m1 = mean(A1, 2);
    m2 = mean(A2, 2);
    m3 = mean(A3, 2);
    mOverall = (m1 + m2 + m3)./3;

    Sw = 0;
    for i = 1:n
        Sw = Sw + (A1(:, i) - m1)*(A1(:, i) - m1)';
    end
    for i = 1:n
        Sw = Sw + (A2(:, i) - m2)*(A2(:, i) - m2)';
    end
    for i = 1:n
        Sw = Sw + (A3(:, i) - m3)*(A3(:, i) - m3)';
    end
```

```
    SB = ((m1 - mOverall)*(m1 - mOverall)' + (m2 - mOverall)*...
        (m2 - mOverall)' + (m3 - mOverall)*(m3 - mOverall)')/3;

    [V2, D] = eig(SB, Sw);
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w, 2);

    v1 = w'*A1;
    v2 = w'*A2;
    v3 = w'*A3;

    vM1 = mean(v1);
    vM2 = mean(v2);
    vM3 = mean(v3);

    sort1 = sort(v1);
    sort2 = sort(v2);
    sort3 = sort(v3);

   % v3 < threshold1 < v1 < threshold2 < v2

    bot = length(sort3);
    top = 1;
    while sort3(bot) > sort1(top)
        bot = bot - 1;
        top = top + 1;
    end
    threshold12 = (sort3(bot) + sort1(top))/2;

    bot = length(sort1);
    top = 1;
    while sort1(bot) > sort2(top)
        bot = bot - 1;
        top = top + 1;
    end
    threshold23 = (sort1(bot) + sort2(top))/2;
end
```

```
  Columns 1 through 19

   1   1   1   1   1   1   1   0   0   0   0   0   0   1   1   1   0   0   0

  Columns 20 through 38

   0   1   1   1   1   1   1   1   1   1   1   1   1   1   0   1   1   1   1

  Columns 39 through 57

   1   1   0   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```

```
    Columns 58 through 60

     1   1   1

      0.8000
```

Published with MATLAB® R2019b

Contents


```
clc; clear all; close all;
```

Load music and compute spectrogram

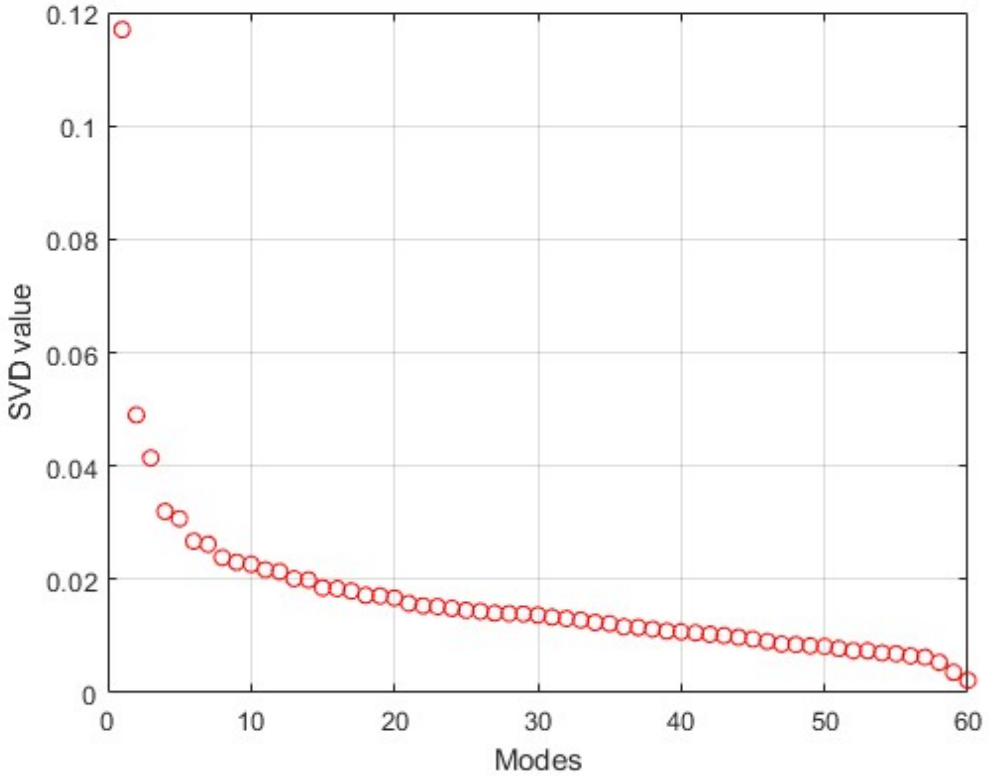
```
pathTrain = 'songs\test2\train\*.mp3';
nClips = 10;
clipLength = 5;
[Train, fNameTrain] = spectro(pathTrain, nClips, clipLength);
```

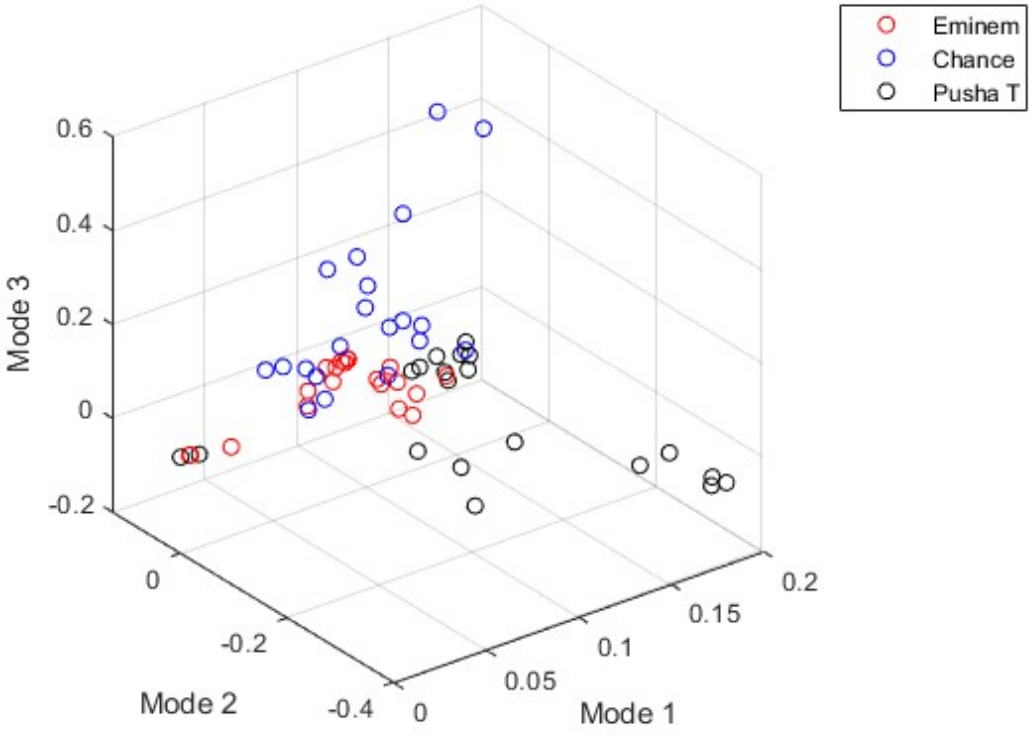
Perform SVD on the spectrogram

```
[U, S, V] = svd(Train, 'econ');
diagS = diag(S)/sum(diag(S));

figure(1)
plot(diagS, 'ro');
grid on;
xlabel('Modes');
ylabel('SVD value');

figure(2)
plot3(V(1:(2*nClips), 1), V(1:(2*nClips), 2), V(1:(2*nClips), 3), 'ro');
grid on;
hold on;
plot3(V((2*nClips + 1):(4*nClips), 1), V((2*nClips + 1):(4*nClips), 2), ...
    V((2*nClips + 1):(4*nClips), 3), 'bo');
plot3(V((4*nClips + 1):(6*nClips), 1), V((4*nClips + 1):(6*nClips), 2), ...
    V((4*nClips + 1):(6*nClips), 3), 'ko');
legend('Eminem', 'Chance', 'Pusha T')
xlabel('Mode 1');
ylabel('Mode 2');
zlabel('Mode 3');
```


SVD value
Modes
0.12
0.1
0.08
0.06
0.04
0.02
0
0
10
20
30
40
50
60


Eminem
Chance
Pusha T
Mode 3
Mode 2
Mode 1
0.6
0.4
0.2
0
-0.2
0
-0.2
-0.4
0
0.05
0.1
0.15
0.2

Load test

```
pathTest = 'songs\test2\test\*.mp3';
[Test, fNameTest] = spectro(pathTest, nClips, clipLength);
```

Build Classifier

```
feature = 8; % 8
[U1, S1, V1, threshold12, threshold23, w, sort1, sort2, sort3] ...
    = artist_trainer(Train, nClips, feature);

pval = w'*(U1'*Test);

resultL = length(pval);
result = strings(1, resultL);
for i = 1:length(pval)
    pval1 = pval(i);
    if pval1 >= threshold23
        result(i) = "C";
    elseif pval1 >= threshold12
        result(i) = "E";
    else
        result(i) = "P";
    end
end

answer = strings(1, resultL);
for i = 1:length(result)
    if i <= resultL/3
        answer(i) = "E";
    elseif i <= resultL*2/3
        answer(i) = "C";
    else
        answer(i) = "P";
    end
end
compare = result == answer;
disp(compare);
accuracy = length(find(compare == 1))/length(result);
disp(accuracy)

function [Data, fName] = spectro(path, nClips, clipLength)
    folder = dir(path);
    nSongs = length(folder);
    Data = [];
    fName = [];
    for i = 1:nSongs
        fname = folder(i).name;
        fName = [fName, convertCharsToStrings(fname)];
        [y, Fs] = audioread([path(1:end -5), fname]);
        y = (y(:, 1) + y(:, 2))/2;
```

```
        s = y';
        t = (1:length(s))/Fs;
        lengthSong = t(end);
        totalClips = lengthSong/clipLength;
        clipSpace = totalClips/nClips;
        tempData = [];
        for j = 1:clipSpace:totalClips
            sStart = floor((j - 1)*clipLength);
            if j == 1
                sStart = 1;
            end
            sEnd = sStart + clipLength;
            clip = s(1, sStart*Fs : sEnd*Fs);
            clipSpec = abs(spectrogram(clip));
            clipSpec = reshape(clipSpec, ...
                size(clipSpec, 1)*size(clipSpec, 2), 1);
            tempData = [tempData, clipSpec];
        end
        Data = [Data, tempData - mean(tempData(:))];
    end
end

function [U, S, V, threshold12, threshold23, w, sort1, sort2, ...
    sort3] = artist_trainer(Data, nSample, feature)

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

    n = 2*nSample;

    Artist = S*V';
    U = U(:, 1:feature);
    A1 = Artist(1:feature, 1:n);
    A2 = Artist(1:feature, (n + 1):(n + n));
    A3 = Artist(1:feature, (n + n + 1):(n + n + n));

    m1 = mean(A1, 2);
    m2 = mean(A2, 2);
    m3 = mean(A3, 2);
    mOverall = (m1 + m2 + m3)./3;

    Sw = 0;
    for i = 1:n
        Sw = Sw + (A1(:, i) - m1)*(A1(:, i) - m1)';
    end
    for i = 1:n
        Sw = Sw + (A2(:, i) - m2)*(A2(:, i) - m2)';
    end
    for i = 1:n
        Sw = Sw + (A3(:, i) - m3)*(A3(:, i) - m3)';
    end

    SB = ((m1 - mOverall)*(m1 - mOverall)' + (m2 - mOverall)*...
        (m2 - mOverall)' + (m3 - mOverall)*(m3 - mOverall)')/3;
```

```
    [V2, D] = eig(SB, Sw);
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w, 2);

    v1 = w'*A1;
    v2 = w'*A2;
    v3 = w'*A3;

    vM1 = mean(v1);
    vM2 = mean(v2);
    vM3 = mean(v3);

    sort1 = sort(v1);
    sort2 = sort(v2);
    sort3 = sort(v3);

   % v3 < threshold1 < v1 < threshold2 < v2

    bot = length(sort3);
    top = 1;
    while sort3(bot) > sort1(top)
        bot = bot - 1;
        top = top + 1;
    end
    threshold12 = (sort3(bot) + sort1(top))/2;

    bot = length(sort1);
    top = 1;
    while sort1(bot) > sort2(top)
        bot = bot - 1;
        top = top + 1;
    end
    threshold23 = (sort1(bot) + sort2(top))/2;
end
```

```
  Columns 1 through 19

   1   1   1   1   1   1   1   1   1   1   0   0   0   0   0   1   0   0   0

  Columns 20 through 30

   0   0   0   0   1   0   0   1   0   0   0

    0.4333
```

Published with MATLAB® R2019b

Contents


```
clc; clear all; close all;
```

Load music and compute spectrogram

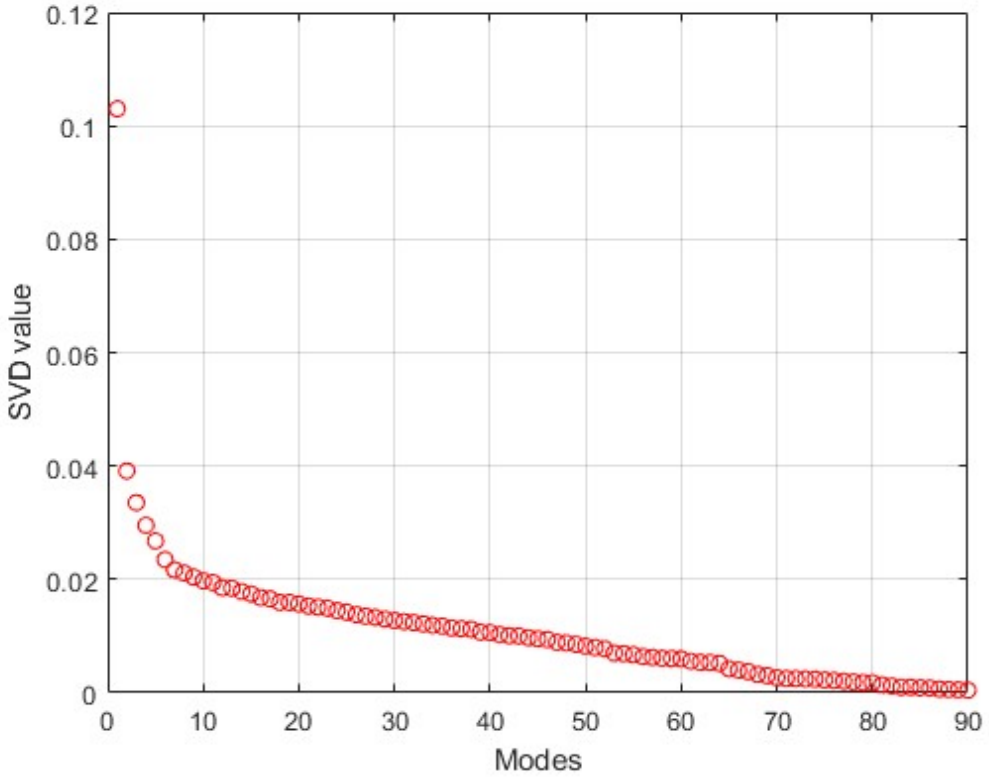
```
pathTrain = 'songs\test3\train\*.mp3';
nClips = 10;
clipLength = 5;
[Train, fNameTrain] = spectro(pathTrain, nClips, clipLength);
```

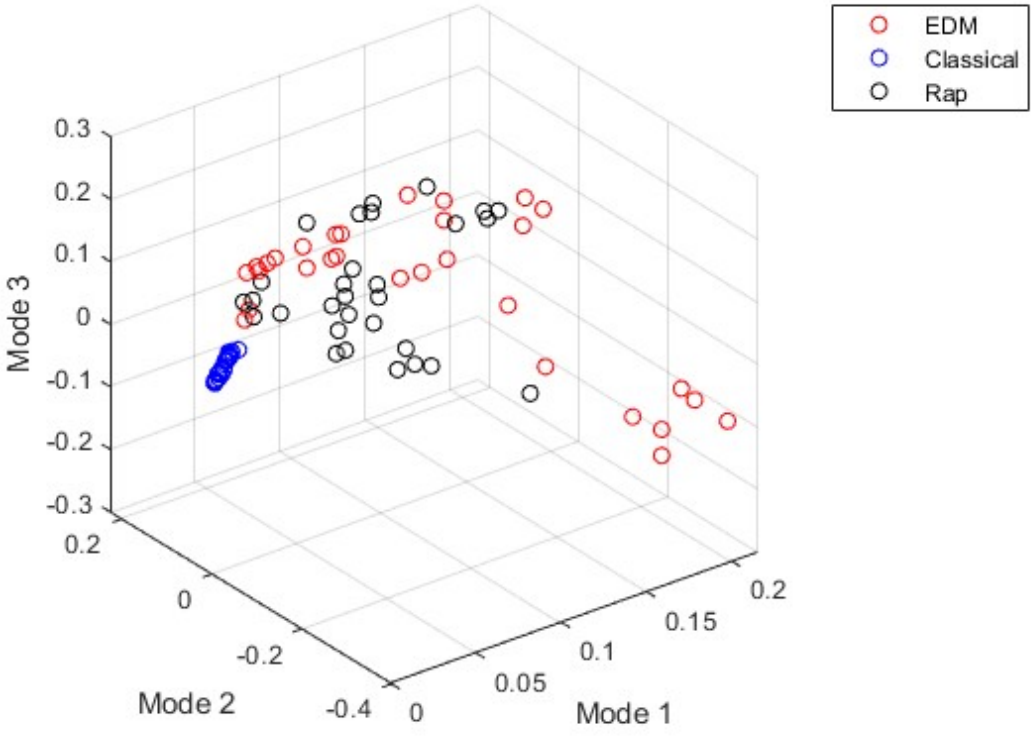
Perform SVD on the spectrogram

```
[U, S, V] = svd(Train, 'econ');
diagS = diag(S)/sum(diag(S));

figure(1)
plot(diagS, 'ro');
grid on;
xlabel('Modes');
ylabel('SVD value');

figure(2)
plot3(V(1:(3*nClips), 1), V(1:(3*nClips), 2), V(1:(3*nClips), 3), 'ro');
grid on;
hold on;
plot3(V((3*nClips + 1):(6*nClips), 1), V((3*nClips + 1):(6*nClips), 2), ...
    V((3*nClips + 1):(6*nClips), 3), 'bo');
plot3(V((6*nClips + 1):(9*nClips), 1), V((6*nClips + 1):(9*nClips), 2), ...
    V((6*nClips + 1):(9*nClips), 3), 'ko');
legend('EDM', 'Classical', 'Rap')
xlabel('Mode 1');
ylabel('Mode 2');
zlabel('Mode 3');
```


SVD value
Modes
0.12
0.1
0.08
0.06
0.04
0.02
0
0
10
20
30
40
50
60
70
80
90


EDM
Classical
Rap
Mode 3
Mode 2
Mode 1
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0.2
0
-0.2
-0.4
0
0.05
0.1
0.15
0.2

Load test

```
pathTest = 'songs\test3\test\*.mp3';
[Test, fNameTest] = spectro(pathTest, 10, clipLength);
```

Build Classifier

```
feature = 3; % 8
[U1, S1, V1, threshold12, threshold23, w, sort1, sort2, sort3] ...
    = genre_trainer(Train, nClips, feature);

pval = w'*(U1'*Test);

resultL = length(pval);
result = strings(1, resultL);
for i = 1:length(pval)
    pval1 = pval(i);

    if pval1 >= threshold23
        result(i) = "EDM";
    elseif pval1 >= threshold12
        result(i) = "RAP";
    else
        result(i) = "CLASS";
    end
end

answer = strings(1, resultL);
for i = 1:length(result)
    if i <= resultL/3
        answer(i) = "EDM";
    elseif i <= resultL*2/3
        answer(i) = "CLASS";
    else
        answer(i) = "RAP";
    end
end
compare = result == answer;
disp(compare);
accuracy = length(find(compare == 1))/length(result);
disp(accuracy)

function [Data, fName] = spectro(path, nClips, clipLength)
    folder = dir(path);
    nSongs = length(folder);
    Data = [];
    fName = [];
    for i = 1:nSongs
        fname = folder(i).name;
        fName = [fName, convertCharsToStrings(fname)];
        [y, Fs] = audioread([path(1:end -5), fname]);
```

```
        y = (y(:, 1) + y(:, 2))/2;
        s = y';
        t = (1:length(s))/Fs;
        lengthSong = t(end);
        totalClips = lengthSong/clipLength;
        clipSpace = totalClips/nClips;
        tempData = [];
        for j = 1:clipSpace:totalClips
            sStart = floor((j - 1)*clipLength);
            if j == 1
                sStart = 1;
            end
            sEnd = sStart + clipLength;
            clip = s(1, sStart*Fs : sEnd*Fs);
            clipSpec = abs(spectrogram(clip));
            clipSpec = reshape(clipSpec, ...
                size(clipSpec, 1)*size(clipSpec, 2), 1);
            tempData = [tempData, clipSpec];
        end
        Data = [Data, tempData - mean(tempData(:))];
    end
end

function [U, S, V, threshold12, threshold23, w, sort1, sort2, ...
    sort3] = genre_trainer(Data, nSample, feature)

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

    n = 3*nSample;

    Genre = S*V';
    U = U(:, 1:feature);
    G1 = Genre(1:feature, 1:n);
    G2 = Genre(1:feature, (n + 1):(n + n));
    G3 = Genre(1:feature, (n + n + 1):(n + n + n));

    m1 = mean(G1, 2);
    m2 = mean(G2, 2);
    m3 = mean(G3, 2);
    mOverall = (m1 + m2 + m3)./3;

    Sw = 0;
    for i = 1:n
        Sw = Sw + (G1(:, i) - m1)*(G1(:, i) - m1)';
    end
    for i = 1:n
        Sw = Sw + (G2(:, i) - m2)*(G2(:, i) - m2)';
    end
    for i = 1:n
        Sw = Sw + (G3(:, i) - m3)*(G3(:, i) - m3)';
    end

    SB = ((m1 - mOverall)*(m1 - mOverall)' + (m2 - mOverall)*...
```

```
        (m2 - mOverall)' + (m3 - mOverall)*(m3 - mOverall)')/3;

    [V2, D] = eig(SB, Sw);
    [~, ind] = max(abs(diag(D)));
    w = V2(:,ind);
    w = w/norm(w, 2);

    v1 = w'*G1;
    v2 = w'*G2;
    v3 = w'*G3;

    vM1 = mean(v1);
    vM2 = mean(v2);
    vM3 = mean(v3);

    sort1 = sort(v1);
    sort2 = sort(v2);
    sort3 = sort(v3);

    % v2 < threshold1 < v3 < threshold2 < v1

    bot = length(sort2);
    top = 1;
    while sort2(bot) > sort3(top)
        bot = bot - 1;
        top = top + 1;
    end
    threshold12 = (sort2(bot) + sort3(top))/2;

    bot = length(sort3);
    top = 1;
    while sort3(bot) > sort1(top)
        bot = bot - 1;
        top = top + 1;
    end
    threshold23 = (sort3(bot) + sort1(top))/2;
end
```

```
  Columns 1 through 19

   0   0   0   0   0   1   1   0   0   1   1   1   1   1   1   1   1   1   1

  Columns 20 through 30

   1   1   1   1   0   1   1   1   1   1   0

    0.7000
```

Published with MATLAB® R2019b