Using PCA and LDA for music classification

Arman Naderi

Abstract:

Data analysis often entails the identification of patterns across a dataset. One method by which data can be analyzed is by classifying data points into unique groups. Using principal component analysis (PCA) and linear discriminate analysis (LDA), an algorithm can be developed to identify unique features of known classes within a given training dataset. This trained algorithm can then be applied to a dataset with unknown classification to sort that data by class. In this paper, I detail the method for the creation of such an algorithm for music classification.

Introduction and Overview:

Classification utilizes principal component analysis (PCA) and linear discriminate analysis (LDA) to separate data into classes based off unique features of that data and those classes. In order to perform classification with a computer algorithm, PCA and LDA must be done on a training dataset in which the identity of the class to which the data belongs to is known. This will inform the identification of thresholds at which the classes differ from one another for the features of interest. Music is a simple application by which classification can be most easily interpreted. I will be training classifiers for three different test cases. The first test case is comparing the classification of music after training from three artists or bands in different genres. The second test case is comparing the classification of music after training from three artists or bands in the same genre. The third test case is comparing the classification of music after training from artists or bands in three different genres. In each test, for each artist, band, or genre, thirty training songs were used. The trained algorithm was then tested on thirty songs for each test. I compared the efficacy of the trained classification algorithm by analyzing its accuracy when classifying the test data.

Algorithm Implementation and Development:

The data was imported as five second long .wav files. For each of the test cases, these music files were downsampled by a factor of 4, and the channel one input is then added to a matrix of music data as a row. Then, a Gabor-transformed spectrogram is applied to each downsampled song; the window width used was 50, and tau was 0.1. This processed music data is then sent for PCA and LDA. First an SVD is performed on a concatenated list of all songs in the test. Then, the algorithm projects the data onto its principal components; a mean of these projections are taken. Within class and between class variance is computed. The maximum eigen value of these variances is then used to inform the creation of its associated eigen vector. All the data is projected onto this eigen vector. Utilizing a sort, the algorithm is able to identify differences between classes by computing a threshold. The threshold is found by identifying the last two points at which each class overlaps. If they do not overlap, the mean of the closest points between each class serves as the threshold.

Computational Results:

The computational results for the algorithm are shown below. Figure 1 shows the results of Test 1. Figure 2 shows the results of Test 2. Figure 3 shows the results of Test 3. The accuracy of test 1 was 0.7. The accuracy of test 2 was 0.333. The accuracy of test 3 was 0.3667.

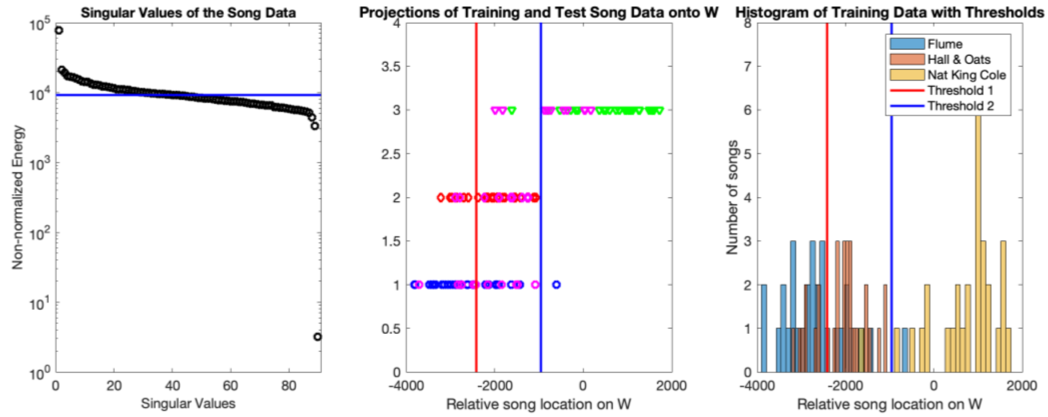

Figure 1: Test 1 results for singular values, projections, and consequent test data histograms, from left to right respectively.

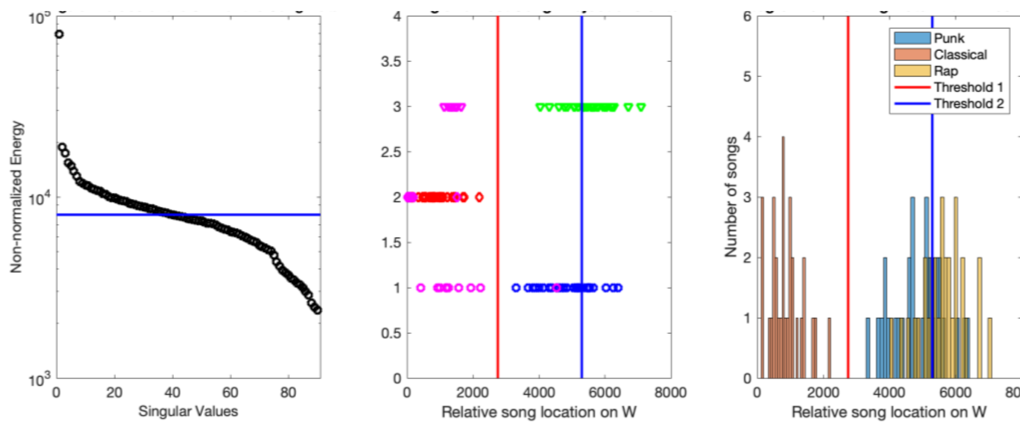

Figure 2: Test 2 results for singular values, projections, and consequent test data histograms, from left to right respectively.

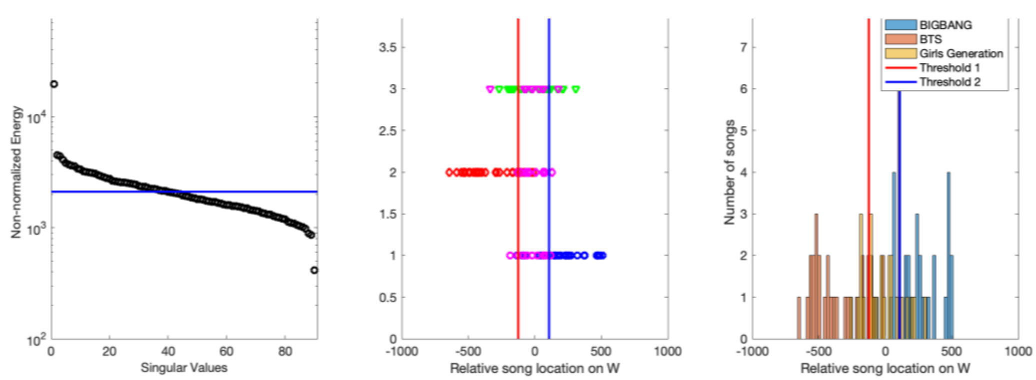

Figure 3: Test 2 results for singular values, projections, and consequent test data histograms, from left to right respectively.

Appendix A:

Appendix B:

```
clear all; close all; clc;

% mc: music classification - band/artist/genre name
% ms: music spectrogram

%% Process music data by creating a Gabor transformed spectrogram
num_song = 30; % number of songs in the data set for a specific mc
ds = 4; % downsample factor
fs = 44100; % sampling frequency of recorded music
fs_d = 44100/ds; %corrected sampling frequency after downsample
L = 5; % length of music is 5 seconds for all clips
tslide=0:0.1:L; % tau = 0.1
n = fs_d * L; % Number of Fourier modes (2^n) (data points)
t2=linspace(0,L,n+1); % Define the time domain discretization
t=t2(1:n); % Only consider the first n points for periodicity
% Fourier components rescaled to have frequency in Hz
k=(1/L)*[0:(n-1)/2 (-n+1)/2:-1]; % frequencies when n is odd
ks=fftshift(k); % Fourier components with zero at the center

%% Test 1 - Band classification
[flume_data] = wav_compiler('flume', num_song, ds);
[hall_data] = wav_compiler('hall', num_song, ds);
[nat_data] = wav_compiler('nat', num_song, ds);
[test1_data] = wav_compiler('test1', num_song, ds);

%% Music spectrogram data matrix for flume, hall, nat
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
% t = discrete time domain
[flume_ms] = spec(flume_data, 50, 0.1, L, n, num_song, t);
[hall_ms] = spec(hall_data, 50, 0.1, L, n, num_song, t);
[nat_ms] = spec(nat_data, 50, 0.1, L, n, num_song, t);
[test1_ms] = spec(test1_data, 50, 0.1, L, n, num_song, t);

%% SVD and LDA applied through trainer function
[U,S,V,w,threshold_1,threshold_2,sort_mc_1,sort_mc_2,sort_mc_3,~,~,~] =
trainer(flume_ms,hall_ms,nat_ms,40);

%% Classify test and Plots

% Plot singular values
figure(1)
subplot(1,3,1)
semilogy(diag(S),'ko','Linewidth',2)
hold on
diag_s = diag(S);
plot([0 length(diag(S))+1],[diag_s(40), diag_s(40)],'b','LineWidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the Song Data','Fontsize',12)
xlabel('Singular Values','Fontsize',12)
ylabel('Non-normalized Energy','Fontsize',12)
```

```
% Plot histogram of training data with thresholds
subplot(1,3,3)
histogram(sort_mc_1,length(sort_mc_1))
hold on
histogram(sort_mc_2,length(sort_mc_2))
histogram(sort_mc_3,length(sort_mc_3))
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
title('Histogram of Training Data with Thresholds','Fontsize',12)
legend('Flume', 'Hall & Oats','Nat King Cole','Threshold 1','Threshold 2')
xlabel('Relative song location on W')
ylabel('Number of songs')
set(gca,'Fontsize',12)

% Classify
TestNum = size(test1_ms,2); % 30
TestMat = U'*test1_ms;  % PCA projection
pval = w'*TestMat;  % LDA projection

% Plot test songs onto song data projections onto w
subplot(1,3,2)
plot(sort_mc_1,ones(length(sort_mc_1)),'ob','Linewidth',2)
hold on
plot(pval(1:10),ones(length(pval(1:10))),'om','Linewidth',2)
plot(sort_mc_2,2.*ones(length(sort_mc_2)),'dr','Linewidth',2)
plot(pval(11:20),2.*ones(length(pval(11:20))),'dm','Linewidth',2)
plot(sort_mc_3, 3.*ones(length(sort_mc_2)),'vg','Linewidth',2)
plot(pval(21:30),3.*ones(length(pval(21:30))),'vm','Linewidth',2)
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
ylim([0 4])
title('Projections of Training and Test Song Data onto W')
xlabel('Relative song location on W')
set(gca,'Fontsize',12)

% Find accuracy
flume_test = pval(1:10);
hall_test = pval(11:20);
nat_test = pval(21:30);

% Accuracy of Nat King Cole
true_high = 0;
for i = 1:10
    if nat_test(i)>threshold_2
        true_high = true_high + 1;
    end
end
accuracy_nat = true_high/length(nat_test);

% Accuracy of Hall & Oats
true_mid = 0;
for i = 1:10
    if hall_test(i)>threshold_1 && hall_test(i)<threshold_2
        true_mid = true_mid + 1;
    end
end
```

```
accuracy_hall = true_mid/length(hall_test);

% Accuracy of Flume
true_low = 0;
for i = 1:10
    if flume_test(i)<threshold_1
        true_low = true_low + 1;
    end
end
accuracy_flume = true_low/length(flume_test);

accuracy_total = (accuracy_nat + accuracy_hall + accuracy_flume)/3;

%% Test 2 - The Case for Seattle

[bb_data] = wav_compiler('bb', num_song, ds);
[bts_data] = wav_compiler('bts', num_song, ds);
[gg_data] = wav_compiler('gg', num_song, ds);
[test2_data] = wav_compiler('test2', num_song, ds);

%% Spectrogram data matrix for flume, hall, nat
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
% t = discrete time domain
[bb_ms] = spec(bb_data, 50, 0.1, L, n, num_song, t);
[bts_ms] = spec(bts_data, 50, 0.1, L, n, num_song, t);
[gg_ms] = spec(gg_data, 50, 0.1, L, n, num_song, t);
[test2_ms] = spec(test2_data, 50, 0.1, L, n, num_song, t);

%% SVD and LDA applied through trainer function
[U,S,V,w,threshold_1,threshold_2,sort_mc_1,sort_mc_2,sort_mc_3,~,~,~] =
trainer(bb_ms,bts_ms,gg_ms,40);

%% Classify and Plots
% Plot singular values
figure(1)
subplot(1,3,1)
semilogy(diag(S),'ko','Linewidth',2)
hold on
diag_s = diag(S);
plot([0 length(diag(S))+1],[diag_s(40), diag_s(40)],'b','LineWidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the SVD of the Song Data','Fontsize',12)
xlabel('Singular Values','Fontsize',12)
ylabel('Non-normalized Energy','Fontsize',12)

% Plot histogram of training data with thresholds
subplot(1,3,3)
histogram(sort_mc_1,length(sort_mc_1))
hold on
histogram(sort_mc_2,length(sort_mc_2))
histogram(sort_mc_3,length(sort_mc_3))
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
title('Histogram of Training Data with Thresholds','Fontsize',12)
legend('BIGBANG', 'BTS','Girls Generation','Threshold 1','Threshold 2')
```

```
xlabel('Relative song location on W')
ylabel('Number of songs')
set(gca,'Fontsize',12)

% Classify
TestNum = size(test2_ms,2); % 30
TestMat = U'*test2_ms;  % PCA projection
pval = w'*TestMat;  % LDA projection

% Plot test songs onto song data projections onto w
subplot(1,3,2)
plot(sort_mc_1,ones(length(sort_mc_1)),'ob','Linewidth',2)
hold on
plot(pval(1:10),ones(length(pval(1:10))),'om','Linewidth',2)
plot(sort_mc_2,2.*ones(length(sort_mc_2)),'dr','Linewidth',2)
plot(pval(11:20),2.*ones(length(pval(11:20))),'dm','Linewidth',2)
plot(sort_mc_3, 3.*ones(length(sort_mc_2)),'vg','Linewidth',2)
plot(pval(21:30),3.*ones(length(pval(21:30))),'vm','Linewidth',2)
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
ylim([0 4])
title('Projections of Training and Test Song Data onto W')
xlabel('Relative song location on W')
set(gca,'Fontsize',12)

% Find accuracy
bb_test = pval(1:10);
bts_test = pval(11:20);
gg_test = pval(21:30);

% Accuracy of BIGBANG
true_high = 0;
for i = 1:10
    if bb_test(i)>threshold_2
        true_high = true_high + 1;
    end
end
accuracy_bb = true_high/length(bb_test);

% Accuracy of Girls Generation
true_mid = 0;
for i = 1:10
    if gg_test(i)>threshold_1 && gg_test(i)<threshold_2
        true_mid = true_mid + 1;
    end
end
accuracy_gg = true_mid/length(gg_test);

% Accuracy of BTS
true_low = 0;
for i = 1:10
    if bts_test(i)<threshold_1
        true_low = true_low + 1;
    end
end
accuracy_bts = true_low/length(bts_test);
```

```
accuracy_total = (accuracy_bb + accuracy_gg + accuracy_bts)/3;

%% Test 3 - Genre Classification

[classical_data] = wav_compiler('classical', num_song, ds);
[punk_data] = wav_compiler('punk', num_song, ds);
[rap_data] = wav_compiler('rap', num_song, ds);
[test3_data] = wav_compiler('test3', num_song, ds);

%% Spectrogram data matrix for punk, rock, classical
% a = 50, tau = 0.1, L = music duration, n = number of datapoints
% t = discrete time domain
[classical_ms] = spec(classical_data, 50, 0.1, L, n, num_song, t);
[punk_ms] = spec(punk_data, 50, 0.1, L, n, num_song, t);
[rap_ms] = spec(rap_data, 50, 0.1, L, n, num_song, t);
[test3_ms] = spec(test3_data, 50, 0.1, L, n, num_song, t);

%% SVD and LDA applied through trainer function
[U,S,V,w,threshold_1,threshold_2,sort_mc_1,sort_mc_2,sort_mc_3,~,~,~] =
trainer(punk_ms,classical_ms,rap_ms,40);

%% Classify and Plots
% Plot singular values
figure(1)
subplot(1,3,1)
semilogy(diag(S),'ko','Linewidth',2)
hold on
diag_s = diag(S);
plot([0 length(diag(S))+1],[diag_s(40), diag_s(40)],'b','LineWidth',2)
set(gca,'Xlim',[0,length(diag(S))+1],'Fontsize',12)
title('Singular Values of the SVD of the Song Data','Fontsize',12)
xlabel('Singular Values','Fontsize',12)
ylabel('Non-normalized Energy','Fontsize',12)


% Plot histogram of training data with thresholds
subplot(1,3,3)
histogram(sort_mc_1,length(sort_mc_1))
hold on
histogram(sort_mc_2,length(sort_mc_2))
histogram(sort_mc_3,length(sort_mc_3))
plot([threshold_1 threshold_1],[0 6],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 6],'b','LineWidth',2)
title('Histogram of Training Data with Thresholds','Fontsize',12)
legend('Punk','Classical','Rap','Threshold 1','Threshold 2')
xlabel('Relative song location on W')
ylabel('Number of songs')
set(gca,'Fontsize',12)

% Classify
TestNum = size(test3_ms,2); % 30
TestMat = U'*test3_ms;  % PCA projection
pval = w'*TestMat;  % LDA projection
```

```
% Plot test songs onto song data projections onto w
subplot(1,3,2)
plot(sort_mc_1,ones(length(sort_mc_1)),'ob','Linewidth',2)
hold on
plot(pval(1:10),ones(length(pval(1:10))),'om','Linewidth',2)
plot(sort_mc_2,2.*ones(length(sort_mc_2)),'dr','Linewidth',2)
plot(pval(11:20),2.*ones(length(pval(11:20))),'dm','Linewidth',2)
plot(sort_mc_3, 3.*ones(length(sort_mc_2)),'vg','Linewidth',2)
plot(pval(21:30),3.*ones(length(pval(21:30))),'vm','Linewidth',2)
plot([threshold_1 threshold_1],[0 8],'r','LineWidth',2)
plot([threshold_2 threshold_2],[0 8],'b','LineWidth',2)
ylim([0 4])
title('Projections of Training and Test Song Data onto W')
xlabel('Relative song location on W')
set(gca,'Fontsize',12)

% Find accuracy
punk_test = pval(1:10);
classical_test = pval(11:20);
rap_test = pval(21:30);

% Accuracy of Rap
true_high = 0;
for i = 1:10
    if rap_test(i)>threshold_2
        true_high = true_high + 1;
    end
end
accuracy_rap = true_high/length(rap_test);

% Accuracy of Punk
true_mid = 0;
for i = 1:10
    if punk_test(i)>threshold_1 && punk_test(i)<threshold_2
        true_mid = true_mid + 1;
    end
end
accuracy_punk = true_mid/length(punk_test);

% Accuracy of Classical
true_low = 0;
for i = 1:10
    if classical_test(i)<threshold_1
        true_low = true_low + 1;
    end
end
accuracy_classical = true_low/length(classical_test);

accuracy_total = (accuracy_rap + accuracy_punk + accuracy_classical)/3;

%% Functions

% wav_compiler: for a given music classification (mc) name
% (band/artist/genre), the function takes in the number of 5 second .wav
% files, downsamples it by some factor, only uses the channel one input and
```

```
% then adds it to a matrix of music data as a row.

% num_song = 30 for all data sets in this algorithm
% ds = 4 for all data sets in this algorithm

function [m_data] = wav_compiler(mc_name, num_song, ds)
cell = {};
for n = 1:num_song
    file_name = '%s_%d.wav';
    file_name = sprintf(file_name, mc_name, n);
    [y, ~]= audioread(file_name);
    y = downsample(y, ds);
    cell{n} = y(:, 1); % only use the channel one inputs
end
m_data = [cell{:}]'; % each row is a song
end

% spec: computes and stores in rows the spectrograms (ms) of songs in music
% data, as a function of window width and tau.

% a = 50, tau = 0.1, L = music duration (5)
% n = number of datapoints (5*Fs/ds), t = discretized time domain

function [ms] = spec(data, width, tau, L, n, num_song, t)
    a = width; % window width
    tslide=0:tau:L;
    ms = []; % store music spectrograms
    m_gabor_s = zeros(length(tslide),n); % store filtered, shifted frequency
data
    for song = 1:num_song
        % Gabor to spectrogram
        for j=1:length(tslide)
            gabor = exp(-a*(t-tslide(j)).^2); % translation parameter
            m_gabor = gabor.*data(song,:); % apply filter to signal
            m_gabor_f = fft(m_gabor);
            m_gabor_s(j,:) = fftshift(abs(m_gabor_f)); % not scaled to stay
in Hz
        end

        row_vec = []; % store in row vectors
        for i = 1:length(tslide)
            row_vec = [row_vec m_gabor_s(i, :)];
        end
        ms =[ms; row_vec];
    end
    ms = ms';
end

% trainer: Computes the thresholds between three different mc's by
% differentiating them by a certain number of features. Computes the SVD of
% the concatenated mc spectrogram data, then performs LDA. After projection
% onto principal components, class mean, variance (within and between)
% are found, and the maximum eigen value and its associated eigenvector are
% computed. After sorting the classes, identify the two points at which
% they overlap; this serves as the threshold. If they do not overlap, take
```

```
% the mean of the closest points between each class; this serves as the
% threshold.

% mc_ms_n = mc music spectrogram data, feature = number of principal
components

function
[U,S,V,w,threshold_1,threshold_2,sort_mc_1,sort_mc_2,sort_mc_3,sorted_high,so
rted_mid,sorted_low] = trainer(mc_ms_1,mc_ms_2,mc_ms_3,feature)
    % number of columns in each spectrogram data set
    n1 = size(mc_ms_1,2);  n2 = size(mc_ms_2,2); n3 = size(mc_ms_3,2);

    % SVD
    % data matrix = n x (3 * num_song)
    [U,S,V] = svd([mc_ms_1, mc_ms_2, mc_ms_3],'econ');
    % U = 2811375x90
    % S = 90x90
    % V = 90x90

    % LDA
    mc_proj = S*V'; % projection onto principal components
    U = U(:,1:feature);
    % U = 2811375xfeature
    mc_proj_1 = mc_proj(1:feature,1:n1);
    mc_proj_2 = mc_proj(1:feature,n1+1:n1+n2);
    mc_proj_3 = mc_proj(1:feature,n1+n2+1:n1+n2+n3);

    m1 = mean(mc_proj_1,2); % (10x1) mean of all columns for each row
    m2 = mean(mc_proj_2,2);
    m3 = mean(mc_proj_3,2);

    Sw = 0; % within class variances
    for k=1:n1 %(30)
        Sw = Sw + (mc_proj_1(:,k)-m1)*(mc_proj_1(:,k)-m1)'; % sigma * sigma =
variance
    end
    for k=1:n2
        Sw = Sw + (mc_proj_2(:,k)-m2)*(mc_proj_2(:,k)-m2)';
    end
    for k=1:n3
        Sw = Sw + (mc_proj_3(:,k)-m3)*(mc_proj_3(:,k)-m3)';
    end

    num_class = 3;
    % m_total = mean([mc_proj_1, mc_proj_2, mc_proj_3]);
    m_total = (m1+m2+m3)/3;
    Sb1 = (m1-m_total)*(m1-m_total)'; % between class
    Sb2 = (m2-m_total)*(m2-m_total)'; % between class
    Sb3 = (m3-m_total)*(m3-m_total)'; % between class
    Sb = (Sb1+Sb2+Sb3)/num_class;

    [V2,D] = eig(Sb,Sw); % linear discriminant analysis
    [~,ind] = max(abs(diag(D)));
    w = V2(:,ind); % maximum eigenvalue and its associated eigenvector
    w = w/norm(w,2);
```

```
    v_proj_1 = w'*mc_proj_1;
    v_proj_2 = w'*mc_proj_2;
    v_proj_3 = w'*mc_proj_3;

    sort_mc_1 = sort(v_proj_1);
    sort_mc_2 = sort(v_proj_2);
    sort_mc_3 = sort(v_proj_3);

    sort_mean_1 = mean(sort_mc_1);
    sort_mean_2 = mean(sort_mc_2);
    sort_mean_3 = mean(sort_mc_3);

    [~, sort_mean_ind] = sort([sort_mean_1, sort_mean_2, sort_mean_3]);
    sorted_high_ind = sort_mean_ind(3);
    sorted_mid_ind = sort_mean_ind(2);
    sorted_low_ind = sort_mean_ind(1);

    sort_mc = [sort_mc_1; sort_mc_2; sort_mc_3];
    sorted_high = sort_mc(sorted_high_ind,:);
    sorted_mid = sort_mc(sorted_mid_ind,:);
    sorted_low = sort_mc(sorted_low_ind,:);

    t1 = length(sorted_low);
    t2 = 1;
    while sorted_low(t1)>sorted_mid(t2)
        t1 = t1-1;
        t2 = t2+1;
    end
    threshold_1 = (sorted_low(t1)+sorted_mid(t2))/2;

    t2 = length(sorted_mid);
    t3 = 1;
    while sorted_mid(t2)>sorted_high(t3)
        t2 = t2-1;
        t3 = t3+1;
    end
    threshold_2 = (sorted_mid(t2)+sorted_high(t3))/2;
end
```