Using Linear Discriminant Analysis to Build a Tertiary Classifier for Music

Jeffrey M Abraham

ABSTRACT

The purpose of this report is use wavelet transforms, principle component analysis, and linear discriminant analysis to train and test a classification model using 5 second clips from different artists and genres.

Keywords: PCA, LDA, Wavelet Transform

I INTRODUCTION AND OVERVIEW

This analysis is being done to classify audio files by extracting the important frequency information using wavelet transforms, identifying principle components that represent large amounts of matrix energy and projecting onto these components to discriminate between categories of song. I train the model for three tests to see how it can preform given certain training data. The first test is to classify three artists from three different genres, the second test is to classify three artists from the same genre, and the third test to classify music from a variety of artists within three distinct genres.

II THEORETICAL BACKGROUND

Principle Component Analysis (PCA)

Principle component analysis uses singular value decomposition(SVD) to break a matrix into three matrices, left singular vectors (U), singular values (Σ) and right singular vectors (V). SVD is derived from an eigenvalue decomposition of the input co-variance matrix, but can be done on any size matrix while eigenvalue decomposition can only be done on square matrices. The input matrix can be represented as $U * \Sigma * V^*$ where V^* is the conjugate transpose of V. It can be helpful to have your input matrix have row mean values of 0. Especially for data with little variation relative to non normalized mean. Singular values σ_n represent coefficients for magnitude of variance along left singular vector V_n.

Linear Discriminant Analysis (LDA)

The concept of linear discriminant analysis is to pick a basis for projection that minimizes the variance within each class while maximizing the variance between classes. This way your data for each class is tightly grouped each class is as spread out as possible. Finding this basis is done using an eigenvalue decomposition using scatter matrices generated as shown in (1) and (2) where X is each column of each within class matrix and n is the total number of classes. Diagonal matrix D and eigenvectors V are generated using S_B*V = S_W*V*D. The eigenvectors V that correspond to the largest eigenvalues maximize $\frac{S_B}{S_W}$ which maximizes the between class variance and minimizes the within class variance. The number of principle components V can be referred to as features. By only using some of these 'features' you can hang on to information that is useful without having your analysis clouded by irrelevant information. More information on how this is done in later sections.

$$S_B = \sum_{i=1}^{n} (\vec{\mu}_i - \vec{\mu})(\vec{\mu}_i - \vec{\mu})^T \tag{1}$$

$$S_W = \sum_{j=1}^{n} \sum_{\vec{X}} (\vec{X} - \vec{\mu}_j)(\vec{X} - \vec{\mu}_j)^T \tag{2}$$

Wavelet Transforms

The wavelet transform is a method of signal decomposition that works by first fitting the function to a series of high frequency periodic functions each with successively different coefficients. This approximation is

then subtracted from the signal and the remainder is fit to a lower frequency. Matlabs continuous wavelet transform function (cwt) by default uses the morse wavelet which has a similar shape to a Gaussian function and uses two parameters γ, the symmetry parameter and P^2, the time-bandwidth parameter. Large time-bandwidth values give large spreads in time and narrow spreads in frequency. γ equal to 3 gives a symmetric wavelet in the frequency domain, while greater values give skew positive wavelets and lesser values give skew negative wavelets. An equation for the morse wavelet in the frequency domain is shown in (3). The term $U(\omega)$ is a unit step function and $a_{P,\gamma}$ is a normalizing constant.

$$\Psi(\omega)_{P,\gamma} = U(\omega) a_{P,\gamma} \omega^{\frac{P^2}{\gamma}} e^{-\omega\gamma} \tag{3}$$

Cross Validation

Cross validation is a method for checking accuracy of a predictive model. It requires that you split your data randomly into training and test data. Standard training:test splits are 70:30 and 80:20. Then you can run the test data through the predictive model and see how accurate your model is. This is important because it is possible to fit a model so that it works perfectly for the training data, but as soon as you test your model you get bad results. This is called over-fitting, and cross validation is an effective method for testing the fit of a model.

III ALGORITHM IMPLEMENTATION AND DEVELOPMENT

Data Creation

The model training and testing algorithms rely on having the data for each class in a matrix with a mono signal for each song in each column. The full data creation script can be found in Appendix B. Two five second long clips were taken from each song from ten seconds to 15 seconds and from 15 seconds to 20 seconds. This was done to try and eliminate the impact of intro length on the audio clips used, and to get twice as much data from each song. To reduce the size of the data files I reduced the sampling frequency by a factor of four by picking out every fourth point. With an initial sampling frequency of 44,100, going to a sampling frequency of 11,025 limits the maximum frequency measurable to Fs/2 or around 5,500Hz. Humans are most sensitive to frequencies within the 20Hz to 5000Hz range, so 25% sampling frequency shouldn't lose any information represented in a song. All test data is loaded into one matrix and the order of each class noted for cross validation.

Creating Spectrograms

I created a function, specwavelet(signal,Fs) to obtain a spectrogram for each clip and reshape it into a column vector. I limited the frequencies represented in the spectrogram to between 40Hz and 6000Hz because that represents most of the frequencies present in music, and also drastically reduces the size of the data files returned.

Model Training Function

To train the model the three training sets are combined into one array, and singular value decomposition is done. The right singular vectors are weighted by their singular values to represent the signal. The U matrix is limited to a certain number of columns as specified in the functions input. The same is done to the rows of the signal array. This limits the number of dimensions that are included in the principle component analysis. LDA is then done for three dimensions, and the two most influential eigenvectors are used for projection. The code for this function can be seen in Appendix B.

Classification

After training the model and vectors of projected signals returned, the mean across each basis vector is calculated. The test data has PCA and LDA done, and their position in each basis vector subtracted from each mean. The two norm of each difference is found and put into a vector. The minimum of this vector is taken and the index recorded. These indices correspond to the class that that test signal is closest to. These are assembled into a row vector and compared using logical operators to a row vector of the actual class that each test signal belongs to. This returns a matrix with ones for every correct prediction and zeros for every incorrect prediction. The number of non-zero entries was found and the total number of test signals divided to give the accuracy.

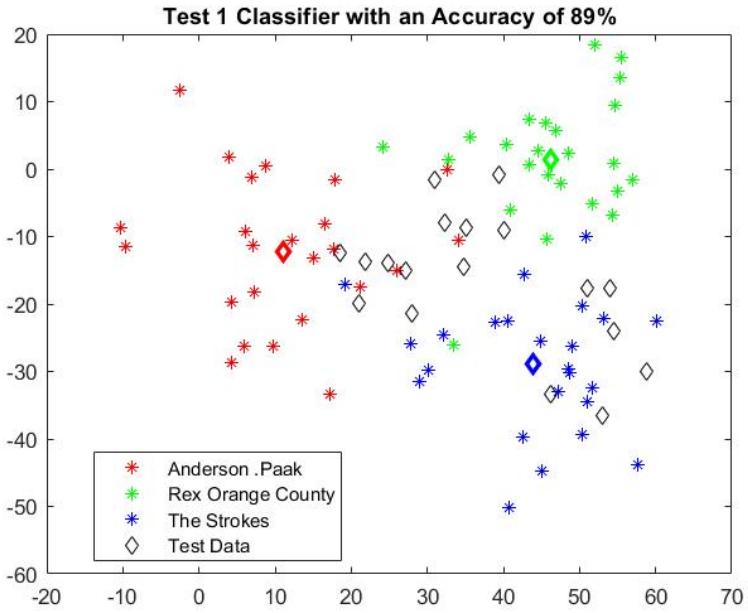

Figure 1. Test 1 LDA Projections

IV COMPUTATIONAL RESULTS

Test 1

For the first test I tried to differentiate between three artists in different genres Anderson .Paak, Rex Orange County and The Strokes. I trained the model with twelve songs by each artist and tested with three songs from each artist. This gave a total training data set of 72 clips (two per song), and a test data set of eighteen clips. This gave a split of 80:20. The plot of all training and test data projected onto the basis for my LDA is shown in Figure 1. The thicker colored diamonds are the mean values for each class. This test yielded an accuracy of 89% or 16 of 18 clips classified correctly.

Test 2

For the second test I chose three artists within the same genre, Pink Floyd, Tame Impala, and The Beatles; all psychedelic rock artists. I trained the model with 20 songs by each artist and tested with three songs from each artist. This gave a total training data set of 120 clips (two per song), and a test data set of 30 clips. This gave a split of 80:20. The plot of all training and test data projected onto the basis for my LDA is shown in Figure 2. This test yielded an accuracy of 60% or 18 of 30 clips classified correctly.

Test 3

For the second test I chose three genres, EDM, Rap, and Classical music. I trained the model with 16 songs by each artist and tested with three songs from each artist. This gave a total training data set of 96 clips (two per song), and a test data set of 24 clips. This gave a split of 80:20. The plot of all training and test data projected onto the basis for my LDA is shown in Figure 3. This test yielded an accuracy of 71% or 17 of 24 clips classified correctly.

V SUMMARY AND CONCLUSIONS

The classifier developed through this exercises works well when given a large amount of representative data. The Second Test preformed poorly in comparison to the others which I attribute to the extreme similarities between the sounds of the bands. The projection of the training set in Figure 2 shows a large within class variance while the other test show tighter clusters. Better results may have been achieved by using a different method of classification than minimum distance to mean. In the end though, I have made

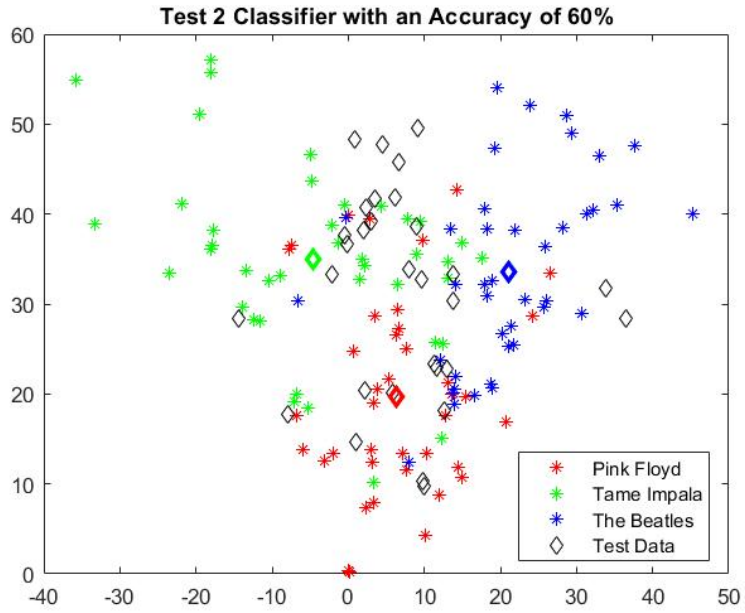

Figure 2. Test 2 LDA Projections

a classifier that can differentiate between songs with better accuracy than guessing in all cases, and with the right data can classify quite accurately.

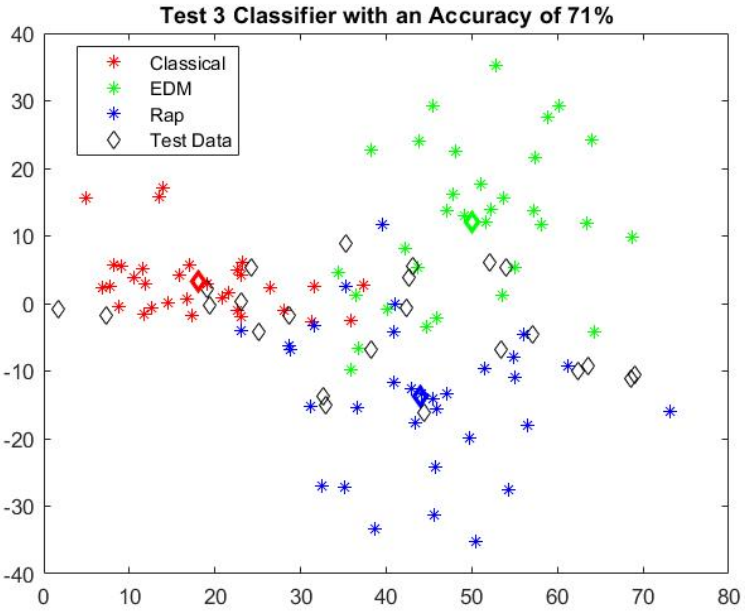

Figure 3. Test 3 LDA Projections

APPENDIX A.

Matlab Functions

nnz(X)
Returns the number of non-zero entries in X

dir('my folder')
Returns a structure array with the information about files contained within my folder

cwt(X,fs,'FrequencyLimits',[f_{min}, f_{max}])
Returns the continuous wavelet transform of X with frequencies scaled to Hz using sampling frequency fs.

APPENDIX B.

Data Making Code

```
clc; clear all; close all;
list1 = dir('Test 1/Anderson .Paak/*.wav');
data1 = [];
paak = zeros(220501,2*length(list1));
for k = 1:length(list1)
    [data1,Fs] = audioread(append('Test 1/Anderson .Paak/', list1(k).
        name));
    data2 = data1(15*Fs:20*Fs);
    data1 = data1(10*Fs:15*Fs);
    paak(:,k) = data1;
    paak(:,12+k) = data2;
end
list2 = dir('Test 1/Rex Orange County/*.wav');
rex = zeros(220501,2*length(list2));
for k = 1:length(list2)
    [data1,Fs] = audioread(append('Test 1/Rex Orange County/', list2(
        k).name));
    data2 = data1(15*Fs:20*Fs);
    data1 = data1(10*Fs:15*Fs);
    rex(:,k) = data1;
    rex(:,12+k) = data2;
end
list3 = dir('Test 1/The Strokes/*.wav');
strokes = zeros(220501,2*length(list3));
for k = 1:length(list3)
    [data1,Fs] = audioread(append('Test 1/The Strokes/', list3(k).
        name));
    data2 = data1(15*Fs:20*Fs);
    data1 = data1(10*Fs:15*Fs);
    strokes(:,k) = data1;
    strokes(:,12+k) = data2;
end
y = [paak rex strokes];
s = zeros((size(y,1)-1)/4,size(y,2));
for l = 1:(size(y,1)-1)/4
    s(l,:) = y(4*l,:);
end
paak = s(:,1:24);
rex = s(:,25:48);
strokes = s(:,49:72);

%%
test = dir('Test 1/Test/*.wav');
tdata = [];
testdata1 = zeros(220501,length(test)*2);
for k = 1:size(test,1)
    [tdata,Fs] = audioread(append('Test 1/Test/', test(k).name));
    tdata2 = tdata(15*Fs:20*Fs);
    tdata = tdata(10*Fs:15*Fs);
    testdata1(:,k) = tdata;
    testdata1(:,9+k) = tdata2;
end
```

```
testdata1s = zeros((size(testdata1,1)-1)/4,size(testdata1,2));
for l = 1:(size(testdata1,1)-1)/4
    testdata1s(l,:) = testdata1(4*l,:);
end

%%
%Part2
list4 = dir('Test 2/Pink Floyd/*.wav');
clip1 = [];
clip2 = [];
floyd = zeros(220501,2*length(list4));
for k = 1:size(list4,1)
    [clip1 Fs] = audioread(append('Test 2/Pink Floyd/',list4(k).name)
        );
    clip2 = clip1(15*Fs:20*Fs);
    clip1 = clip1(10*Fs:15*Fs);
    floyd(:,k) = clip1;
    floyd(:,20+k) = clip2;
end

list5 =dir('Test 2/Tame Impala/*.wav');
tame = zeros(220501,2*length(list4));
for k = 1:size(list5,1)
    [clip1 Fs] = audioread(append('Test 2/Tame Impala/',list5(k).name
        ));
    clip2 = clip1(15*Fs:20*Fs);
    clip1 = clip1(10*Fs:15*Fs);
    tame(:,k) = clip1;
    tame(:,20+k) = clip2;
end

list6 =dir('Test 2/The Beatles/*.wav');
beatles = zeros(220501,2*length(list4));
for k = 1:size(list6,1)
    [clip1 Fs] = audioread(append('Test 2/The Beatles/',list6(k).name
        ));
    clip2 = clip1(15*Fs:20*Fs);
    clip1 = clip1(10*Fs:15*Fs);
    beatles(:,k) = clip1;
    beatles(:,20+k) = clip2;
end

S2 = [floyd tame beatles];
s2 = zeros((size(S2,1)-1)/4,size(S2,2));
for l = 1:(size(S2,1)-1)/4
    s2(l,:) = S2(4*l,:);
end

floyd = s2(:,1:40); tame = s2(:,41:80); beatles = s2(:,81:120);


%%
test2 = dir('Test 2/Test/*.wav');
tdata = [];
testdata2 = zeros(220501,length(test2)*2);
```

```
for k = 1:size(test2,1)
    [tdata,Fs] = audioread(append('Test 2/Test/', test2(k).name));
    tdata2 = tdata(15*Fs:20*Fs);
    tdata = tdata(10*Fs:15*Fs);
    testdata2(:,k) = tdata;
    testdata2(:,15+k) = tdata2;
end
testdata2s = zeros((size(testdata2,1)-1)/4,size(testdata2,2));
for l = 1:(size(testdata2,1)-1)/4
    testdata2s(l,:) = testdata2(4*l,:);
end
%% part 3
list7 = dir('Test 3/EDM/*.wav');
clip1 = [];
clip2 = [];
edm = zeros(220501,2*length(list7));
for k = 1:size(list7,1)
    [clip1 Fs] = audioread(append('Test 3/EDM/',list7(k).name));
    clip2 = clip1(15*Fs:20*Fs);
    clip1 = clip1(10*Fs:15*Fs);
    edm(:,k) = clip1;
    edm(:,16+k) = clip2;
end

list8 =dir('Test 3/Mumble/*.wav');
rap = zeros(220501,2*length(list8));
for k = 1:size(list8,1)
    [clip1 Fs] = audioread(append('Test 3/Mumble/',list8(k).name));
    clip2 = clip1(15*Fs:20*Fs);
    clip1 = clip1(10*Fs:15*Fs);
    rap(:,k) = clip1;
    rap(:,16+k) = clip2;
end

list9 =dir('Test 3/Classical/*.wav');
class = zeros(220501,2*length(list9));
for k = 1:size(list9,1)
    [clip1 Fs] = audioread(append('Test 3/Classical/',list9(k).name))
        ;
    clip2 = clip1(15*Fs:20*Fs);
    clip1 = clip1(10*Fs:15*Fs);
    class(:,k) = clip1;
    class(:,16+k) = clip2;
end

S2 = [edm rap class];
s2 = zeros((size(S2,1)-1)/4,size(S2,2));
for l = 1:(size(S2,1)-1)/4
    s2(l,:) = S2(4*l,:);
end

edm = s2(:,1:32); rap = s2(:,33:64); class = s2(:,65:96);

%% testdata 3

```

```
test3 = dir('Test 3/Test/*.wav');
tdata = [];
testdata3 = zeros(220501,length(test3)*2);
for k = 1:size(test3,1)
    [tdata,Fs] = audioread(append('Test 3/Test/', test3(k).name));
    tdata2 = tdata(15*Fs:20*Fs);
    tdata = tdata(10*Fs:15*Fs);
    testdata3(:,k) = tdata;
    testdata3(:,12+k) = tdata2;
end
testdata3s = zeros((size(testdata3,1)-1)/4,size(testdata3,2));
for l = 1:(size(testdata3,1)-1)/4
    testdata3s(l,:) = testdata3(4*l,:);
end
```

Spectrogram Creation Function

```
function data = specwavelet(signal,Fs)
    clips = size(signal,2);
    data = [];
    for k = 1:clips
        s = signal(:,k)';
        spec = cwt(s,Fs,'FrequencyLimits',[40 6000]);
        data = [data abs(reshape(spec,[],1))];
    end
end
```

Training Function

```
function [U,S,V,w,v1,v2,v3] = class_train3(train_1,train_2,train_3,
    features,dim)
    n1 = size(train_1,2); n2 = size(train_2,2); n3 = size(train_3,2);

    [U,S,V] = svd([train_1 train_2 train_3],'econ');

    signal = S*V';
    U = U(:,1:features);

    %Category features
    cat1 = signal(1:features,1:n1);
    cat2 = signal(1:features,n1+1:n1+n2);
    cat3 = signal(1:features,n1+n2+1:end);

    m1 = mean(cat1,2); m2 = mean(cat2,2); m3 = mean(cat3,2); %
        Category Means

    %Total Mean
    mt = mean(signal(1:features,:),2);

    %Variance Within Category
    Sw = 0;
    for j = 1:n1
        Sw = Sw + (cat1(:,j)-m1)*(cat1(:,j)-m1)';
    end
    for k = 1:n2
        Sw = Sw + (cat2(:,k)-m2)*(cat2(:,k)-m2)';
    end
    for l = 1:n3
```

```
        Sw = Sw + (cat3(:,1)-m3)*(cat3(:,1)-m3)';
    end

    %Variance Between Categories
    Sb = (m1-mt)*(m1-mt)' + (m2-mt)*(m2-mt)' + (m3-mt)*(m3-mt)';

    %Selecting Principle Components
    [V2,D] = eig(Sb,Sw);
    [M,I] = max(abs(diag(D)));
    [~,I2] = max(abs(diag(D).*diag(D<M)));
    w = [V2(:,I) V2(:,I2)]; w = w(:,1:dim); w = w/norm(w,2);

    %LDA Projection
    v1 = w'*cat1;
    v2 = w'*cat2;
    v3 = w'*cat3;
end
```

Test 1

```
clc; clear all; close all;
load('test1.mat');
load('Testdata1.mat');

%Making Spectrograms
data1 = specwavelet(paak,Fs/4);
data2 = specwavelet(rex,Fs/4);
data3 = specwavelet(strokes,Fs/4);

%% training
[U,S,V,w,v1,v2,v3] = class_train3(data1,data2,data3,20,2);

%% testing
m1 = mean(v1,2); m2 = mean(v2,2); m3 = mean(v3,2);

testwave = specwavelet(testdata1s,Fs/4);
testmat = U'*testwave;
pval = w'*testmat;
answ = [];
for k = 1:size(pval,2)
    vars = [norm(pval(:,k)-m1) norm(pval(:,k)-m2) norm(pval(:,k)-m3)
        ];
    [M,I] = min(vars);
    answ = [answ I];
end
accuracy = nnz(answ==[1 1 1 2 2 2 3 3 3 1 1 1 2 2 2 3 3 3])/length(
    answ);
%% plotting
plot(v1(1,:),v1(2,:),'r*')
hold on;
plot(v2(1,:),v2(2,:),'g*')
plot(v3(1,:),v3(2,:),'b*')
plot(pval(1,:),pval(2,:),'kd')
plot(m1(1),m1(2),'rd','Linewidth',2)
plot(m2(1),m2(2),'gd','Linewidth',2)
plot(m3(1),m3(2),'bd','Linewidth',2)
legend('Anderson .Paak','Rex Orange County','The Strokes','Test Data'
```

```
    ,'Location','Best')
title(['Test 1 Classifier with an Accuracy of ' int2str(round(100*
    accuracy)) '%'])
```

Test 2

```
clc; clear all; close all;
load('test2.mat');
load('Testdata2.mat');

data1 = specwavelet(floyd,Fs/4);
data2 = specwavelet(tame,Fs/4);
data3 = specwavelet(beatles,Fs/4);

%%
[U,S,V,w,v1,v2,v3] = class_train3(data1,data2,data3,25,2);
%%
m1 = mean(v1,2); m2 = mean(v2,2); m3 = mean(v3,2);

testwave = specwavelet(testdata2s,Fs/4);
testmat = U'*testwave;
pval = w'*testmat;
answ = [];
for k = 1:size(pval,2)
    vars = [norm(pval(:,k)-m1) norm(pval(:,k)-m2) norm(pval(:,k)-m3)
        ];
    [M,I] = min(vars);
    answ = [answ I];
end
accuracy = nnz(answ==[3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1
    1 2 2 2 2 2])/length(answ);
%%
plot(v1(1,:),v1(2,:),'r*')
hold on;
plot(v2(1,:),v2(2,:),'g*')
plot(v3(1,:),v3(2,:),'b*')
plot(pval(1,:),pval(2,:),'kd')
plot(m1(1),m1(2),'rd','Linewidth',2)
plot(m2(1),m2(2),'gd','Linewidth',2)
plot(m3(1),m3(2),'bd','Linewidth',2)
legend('Pink Floyd','Tame Impala','The Beatles','Test Data','Location
    ','Best')
title(['Test 2 Classifier with an Accuracy of ' int2str(round(100*
    accuracy)) '%'])
```

Test 3

```
clc; clear all; close all;
load('test3.mat');
load('Testdata3.mat');

data1 = specwavelet(class,Fs/4);
data2 = specwavelet(edm,Fs/4);
data3 = specwavelet(rap,Fs/4);

%%
[U,S,V,w,v1,v2,v3] = class_train3(data1,data2,data3,25,2);

```

```
%%
m1 = mean(v1,2); m2 = mean(v2,2); m3 = mean(v3,2);

testwave = specwavelet(testdata3s,Fs/4);
testmat = U'*testwave;
pval = w'*testmat;
answ = [];
for k = 1:size(pval,2)
    vars = [norm(pval(:,k)-m1) norm(pval(:,k)-m2) norm(pval(:,k)-m3)
        ];
    [M,I] = min(vars);
    answ = [answ I];
end
accuracy = nnz(answ==[1 1 1 1 2 2 2 2 3 3 3 3 1 1 1 1 2 2 2 2 3 3 3
    3])/length(answ);
%%
plot(v1(1,:),v1(2,:),'r*')
hold on;
plot(v2(1,:),v2(2,:),'g*')
plot(v3(1,:),v3(2,:),'b*')
plot(pval(1,:),pval(2,:),'kd')
plot(m1(1),m1(2),'rd','Linewidth',2)
plot(m2(1),m2(2),'gd','Linewidth',2)
plot(m3(1),m3(2),'bd','Linewidth',2)
legend('Classical','EDM','Rap','Test Data','Location','Best')
title(['Test 3 Classifier with an Accuracy of ' int2str(round(100*
    accuracy)) '%'])
```