Elizabeth Chin
Kevin Stangl
Math 156
Spring 2016
Final Project

Music Genre Classification using Soft-SVM and Mel-Frequency Cepstrum Coefficients

PROBLEM

Our goal is to take mp3 files of music and classify the qualitative musical genre, i.e. jazz, versus classical rock, classical, or rap. In the real world, genre classification methods are used for music content retrieval methods and semantic search programs.

This is a difficult problem because categorically similar music has a large amount of variance even within genre. Moreover the ground truth is a human judgment, and thus can often be quite blurry and music can be associated with one category for arbitrary reasons rather than substantive difference. Given past work in the literature we expect to get around an 80-85% correct classification rate.

DATA

We selected 40 of the most popular jazz and classic rock songs; 20 were jazz songs and 20 were classic rock songs. Pieces were selected to be sung by different artists, and all pieces had vocals to reduce the effects of confounding variables. We separated our data randomly into testing and training (training: 10 jazz + 10 classic rock and testing: 10 jazz + 10 classic rock).

METHODS

Overview: Song → Segments → Features Extracted → SVM → Classification

Segmentation:

We first segment the songs by structural components to allow us to better extract information based on the song structure such as segment boundaries, musical form, verse, chorus, ect. We especially wanted to identify repetitive structures to reduce redundancy and non-informative samples in our dataset. We first converted our audiofile into beats using the "VOICEBOX" toolbox[1] to analyze the tempo of the music signal. The beat and audio files then served as input for automated segmentation. The algorithm for segment boundary detection used a self-similarity matrix (autocorrelation matrix) to match a song's frames to itself to determine repetitive motifs of a song. A "novelty score" was derived using a Gaussian Kernel over the self-similarity matrix.

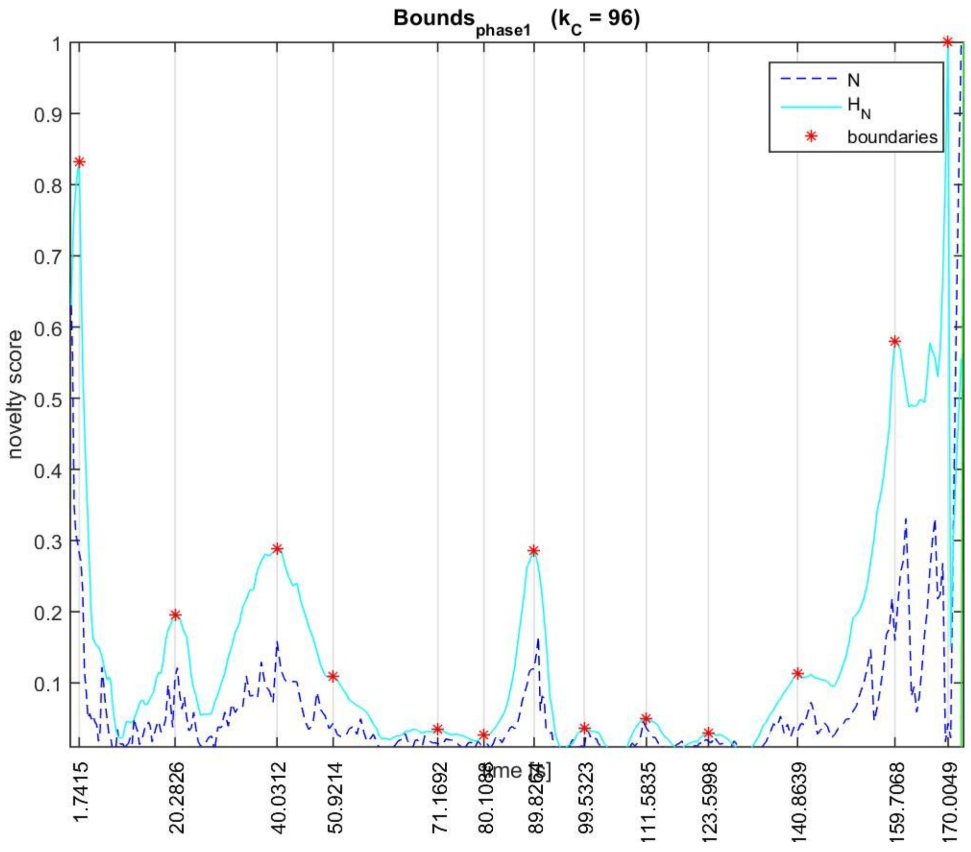
The self-similarity matrix (**S**) is calculated using a Euclidian distance function (d_S):

$$\mathbf{S}(i,j) = d_{\mathbf{S}}(\mathbf{v}_i, \mathbf{v}_j) \quad i,j = 1,\ldots,n$$

Novelty score N, where $\mathbf{C}_k$ denotes a Gaussian tapered kernel of size k_C.

$$N(i) = \sum_{m=-k_C/2}^{k_C/2} \sum_{n=-k_C/2}^{k_C/2} \mathbf{C}_k(m,n)\mathbf{S}(i+m, i+n)$$

After running through the novelty score with a low pass filter to reduce noise, local maxima were set as segment boundaries.[3] Figure 1 is an example of the novelty score segmentation for Jazz music.



Feature Extraction:

The core of this problem is well chosen feature extraction. We decided to focus on Mel Frequency Cepstrum Coefficients (MFCC). MFCCs are short-term spectral decomposition of audio signals that approximates the general frequency characteristics, as well as auditory response of human hearing. MFCCs are commonly used in speech recognition systems. `Cepstrum' is in the reversal of the first 4 letters of the word `spectrum' and it is a measure of the rate of change of different spectrum bands, originally designed to characterize seismic echoes from bombs and earthquakes. The pseudocode for MFCCs are as follows:

1. Take the Fourier Transform within each segment
2. Smooth the frequencies and map the spectrum obtained above onto the mel scale
3. Take the logs of the powers at each of the mel frequencies
4. Take the discrete cosine transform of the list of mel log powers, as if it were a signal

5. The MFCCs are the amplitudes of the resulting spectrum

$$M(f) = 1125 \ln(1 + f/700)$$

The equation above represents the conversion from frequency to MFCCs. We took 12 mel coefficients per data point, then calculated additional characteristics of the signal by looking at energy, delta, and delta-delta. The energy is the log-energy of the MFCCs, which reflects the power of the signal. Since the MFCC vector describes the power spectral envelope over one frame, we also calculated delta and delta-delta coefficients, differential and acceleration respectively, to describe the trajectories of MFCCs over time. The equation is as follows:

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}$$

where d_t is a delta coefficient, from frame t computed in terms of the static coefficients c_{t+N} to c_{t-N}. A typical value for N is 2. Delta-delta (Acceleration) coefficients are calculated in the same way, but they are calculated from the deltas, not the static coefficients (equation and description from citation).[3]

Support Vector Machine:

We seek to classify the data using Support Vector Machines following the methods developed in Xu *et al.* As a proof-of-concept we first develop the model to separate classic rock and jazz (which we believe are relatively dissimilar categories). We adopted a classic training/test split to validate our results. We believe SVM is a good choice since it has been used by successfully by other research groups and works quickly on large datasets. Moreover, our features are designed explicitly for this type of classification.

We first created a training set consisting of 10 rock and 10 jazz songs and a test of the same size. We then segmented the songs and extracted the features by segment.

Crucially, each segmentation generated a very large number of MPC vectors and the number of extracted values was not constant. Given some crude experimentation, we decided to use 500 coefficients from each of the twenty songs so the data matrix we called SVM on was 10,000-by-42 (42 is the dimensionality of the MPC, energy, delta, and delta-delta coefficients).

It is important to note that we trained the model on the individual MPC coefficients. This is a bagging method since we are sub-sampling our data. In effect we input to the model 5000 MPCC coefficients from jazz music and 5000 from classic rock. This very large number of feature vectors seems to be a possible cause of our very high accuracy. (as will be shown in results).

We implemented two different sub-sampling methods. The songs are segmented and each song has a different number of segments. The first method takes the first 500 MPC coefficients of the first segment of each song. The second method picks a random segment from each song. We pick a random segment from each song, and pick the first 500 MPCC from that segment

Our classification method for songs in the test set was to extract the features using the same methods that generated the training samples. We then use the learned SVM classifier on every MPP coefficient (feature) and assign each song to the class provided by a majority vote of the features for that song. Thus each song is assigned to a class using the results from 500 individual SVM classifications. We believe this will provide robust and accurate results since for a song to misclassified at least 250 features would need to be classified incorrectly. Assuming every feature is correctly classified with a probability of p in (0,1), p^(250) is a very small number.

EXPERIMENTS

1a: Train with first 500 MPCC from the first segment of each song using the soft-SVM (same code as we used in Lab #7).
1b: Same as Method 1a but we use random sampling method.
2a: Use the same sampling method as problem 1a. but train the model uses \`fitcsvm' and the built in matlab function to \`predict' that evaluates new data using the trained SVM model.
2b: Use the same training method as Experiment 2a. but using the random sampling method from Experiment 1b.

RESULTS

Test data using the soft SVM from Lab #7 is evaluated using \`evaluate_features_naivesvm.' This function takes each feature vector and calculates X'w+b. The reason the values on the x-axis have such a wide range, [-500,500] is because the classification is generated by the sum of all the features (there is 500 per song. Thus a data point with a value greater than 0 is assigned to Class 1 (classic rock) while a data point less than 0 is assigned to Class -1 (jazz). A first glance shows the high accuracy of our method. The results of these classifications will appear in the Data Section, with the title \`Labels_1a' for the results of the test set from Experiment 1a. The classification rates of the experiments 1a, 1b,2a, 2b are 100%, 95%,100%, and 95% respectively (20/20, 19/20, 20/20, 19/20). We can also possibly interpret data points near the boundary as being similar to other genre.

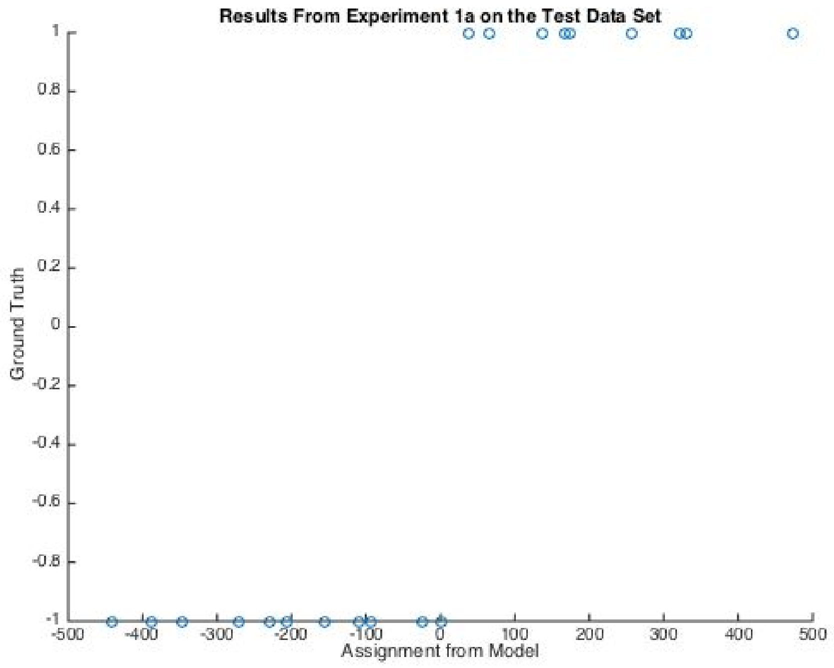


Figure 2a.

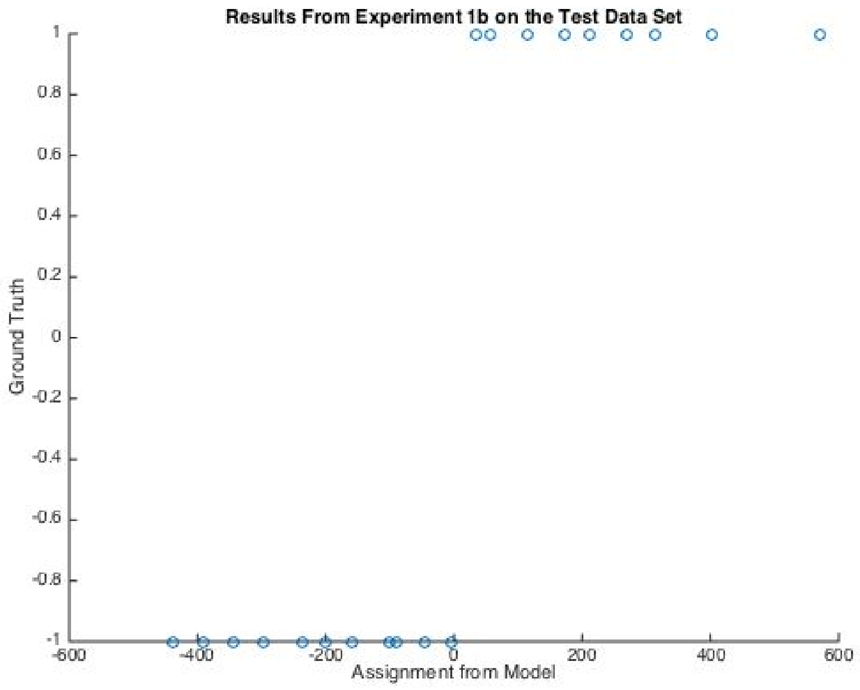


Figure 2b.

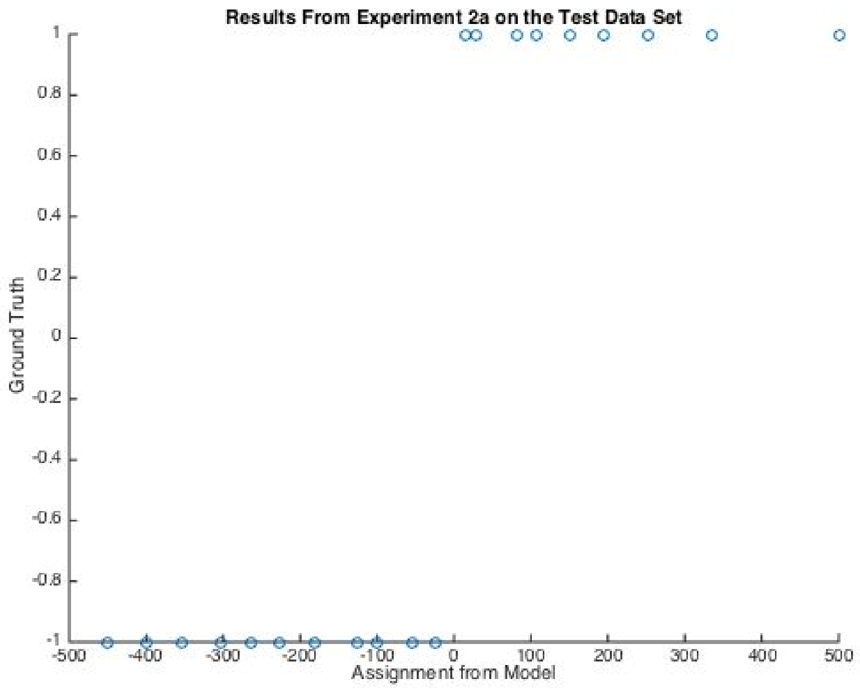


Figure 3a.

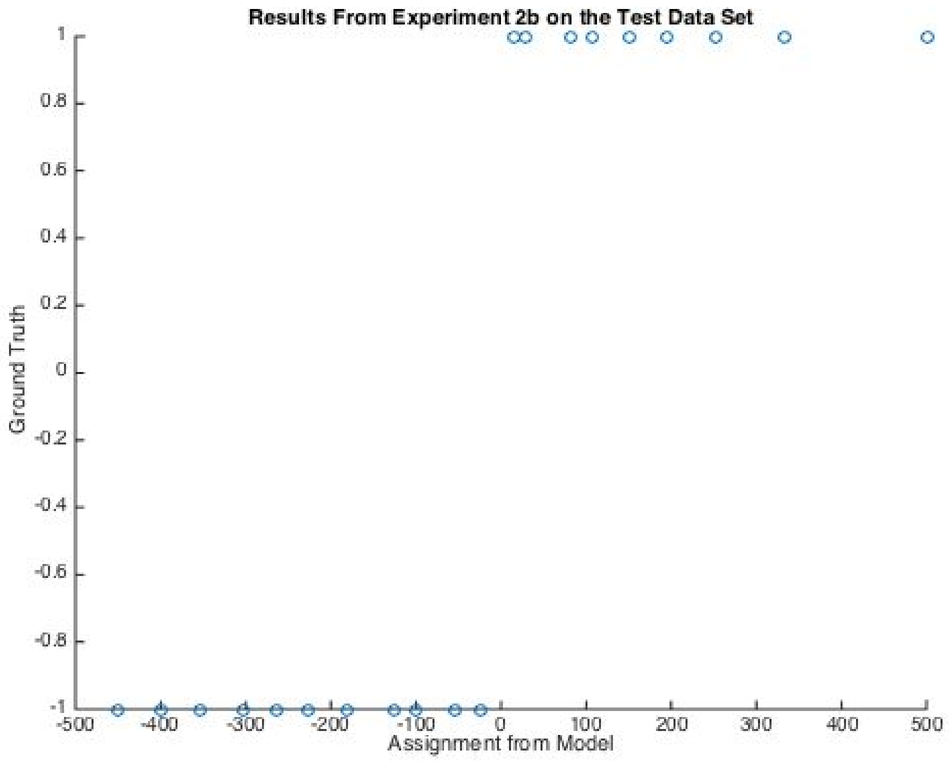


Figure 3b.

Further Work/Conclusions:

These results are remarkably good. While we only validated on a relatively small datasets, the consistency of different methods of SVM makes me feel confident in the accuracy of our approach. This is especially surprising since the other methods in the literature use a number of different features while we use only one. I believe our high degree of accuracy can be associated to the fact that every classification is the sum of five hundred classifications of individual feature vectors from random segments. In effect, it is similar to the ensemble methods we studied in class.

It is additionally surprising that the linear soft-SVM performed strictly better than the MATLAB built in SVM using the non-linear radial basis functions. Additionally, the two sampling methods had little difference.

I find it very strange how the results from the test data are monotonically decreasing and increasing. I did not sort them. This could be some sort of artifact from the segmentation and feature extraction process.

There could be some degree of confounding since the datasets were curated by hand and thus it could be that we picked samples that are relatively divergent by genre than typical songs in these categories or alternatively that we selected songs are relatively similar within categories.

Ideally we like to extend our research to classifying N separate categories using a tree based approach. For the case of four genres, we could use one classifier to separate the music into two pairs of related genre. This classifier would be trained using a dataset that lumps each pair together. Each pair would be further separated using another SVM that was trained on each genre. The downside of this approach would be the increasing number of necessary classifiers.

Code:

Test Results:

The ground truth is the first ten values should be positive; the second ten should be negative. Each value here is the sum of the classifications of 500 feature vectors from each song.

Experiment 1a.

```
473.646682195209
331.143173201309
320.332667426578
257.015402821054
174.859123055442
168.046853432071
136.447386956034
64.7755628319647
37.7260192402618
0.299569136921691
-25.1807062555382
-92.9950018336989
```

-110.312148954301
-156.008430543123
-205.134341188904
-230.204900811278
-271.188117719225
-346.249407723439
-389.227655827814
-442.240635482539

Experiment 1b
571.872520288275
401.541296073457
312.618230971504
269.065579742700
210.770118079754
172.169028665878
115.156709667140
55.4754297791455
35.8541025343158
-5.20310905587371
-45.1364858758049
-90.3321479662641
-100.586395236784
-159.270777545182
-201.800765280503
-235.083471665039
-297.031082706398
-344.746331026784
-391.970264359685
-437.831631350863

Experiment 2a.
500
336
252
194
152
106
82
28
14
-24
-54
-100
-126

-182
-226
-264
-302
-354
-400
-450

Experiment 2b.
500
334
252
194
152
106
82
28
14
-24
-54
-100
-126
-182
-226
-264
-302
-354
-400
-450

References:

1. Brookes, Michael. VOICEBOX: Speech Processing Toolbox for MATLAB.
2. Ewald Peiszer, Thomas Lidy, and Andreas Rauber. Automatic audio segmentation: Segment boundary and structure detection in popular music. In Proceedings of the 2nd International Workshop on Learning the Semantics of Audio Signals (LSAS), 2008.
3. "Mel Frequency Cepstral Coefficient (MFCC) tutorial." Practical Graphy. Web. 03 June 2016.
4. Li, Feng, You You, Yuqin Lu, and Yuqing Pan. "An Automatic Segmentation Method of Popular Music Based on SVM and Self-similarity." *Human Centered Computing Lecture Notes in Computer Science* (2015): 15-25. Web. 1 June 2016.
5. Mandel, Michael I., Graham E. Poliner, and Daniel P. W. Ellis. "Support Vector Machine Active Learning for Music Retrieval." *Multimedia Systems* 12.1 (2006): 3-13. Web.

6. Xu, Changsheng, N.c. Maddage, Xi Shao, Fang Cao, and Qi Tian. "Musical Genre Classification Using Support Vector Machines." *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03).* (n.d.): n. pag. Web.

CODE

Script to call beat generation, segmentation, and feature extraction

```
%% run scripts for music classification. note that you need to do this for both training
and testing
for i=1:10
   fileName = strcat('./segmentation/corpus/clips/j',int2str(i),'.mp3');
   beatFileName = strcat('./segmentation/corpus/beats/j',int2str(i),'.beats');
   davies_standard(fileName,beatFileName); %generate beats
   glob = strcat('j',int2str(i),'.mp3');
   J{i} = filefun(q_alg, [PATH_WAV22050 glob], 'args', args); %generate segments
   mfc{i} = segment(fileName,J{i}); %get mfccs
end

for i=1:10
   fileName = strcat('./segmentation/corpus/clips/cr',int2str(i),'.mp3');
   beatFileName = strcat('./segmentation/corpus/beats/cr',int2str(i),'.beats');
   davies_standard(fileName,beatFileName);
   glob = strcat('cr',int2str(i),'.mp3');
   CR{i} = filefun(q_alg, [PATH_WAV22050 glob], 'args', args); %generate segments
   mfc{i+10} = segment(fileName,CR{i}); %get mfccs
end
```

Script to extract features

```
%% input mp3 (location to mp3 file) and segments (time points of segment boundaries)
%Outputs mfc, energy, delta, and delta-delta coefficients of segments
function [mfc] = segment(mp3,segments)
         [data, Fs] = audioread(mp3);
   data = data(:,1);
   for i = 1:(length(segments)-1)
      d_start = floor(segments(i)*Fs);
      d_end = floor(segments(i+1)*Fs-1);
      dat = data(d_start:d_end);
      mfc{i} = melcepst(dat,Fs,'0EdD',12); %Mel-Frequency Cepstral Coefficients
   end
end
```

davies_standard (beat generation):
http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
melcepst (MFCCs): MIRtoolbox

ep_audiosegmentation (segmentation): not publicly available. If interested, let me know and I can email it to you (thousands of lines of code + heavy edits because of datasets and lack of documentation).

prep_data_for_svm:

```
function [X,t]=prep_data_for_svm(mfc)
%this is just a naive helper function to sample
%some subset of the features and put them in a form
%ready for SVM
t=zeros(10000,1);
t(1:5000)=1;
t(5000:10000)=-1;
X=zeros(10000,42);
%maxindex=[33,6,13,10,3,7,9,1,19,12,5,8,8,1,14,9,16,8,1,11];   %train
maxindex=[7,5,1,1,5,5,11,14,3,12,12,7,10,4,9,1,3,13,1,6];    %test
%takes 500 datapoints from the first segment for the first  song
 for j=1:20 %j is the song
        k=10000000;
        while (k>maxindex(j)) || (k<1)  %used for random sampling
        k=round(rand*maxindex(j));
        end
     for i=1:500
        X(j*i,:)=mfc{1,j}{1,k}(i,:);
        % X(j*i,:)=mfc{1,j}{1,1}(i,:); %used for nonrandom sampling
     end
 end
%Used to train the model using built in matlab svm with radial basis function
function [ svmmodel ] =matlabsvm(X_train, t_train)
svmmodel=fitcsvm(X_train,t_train,'KernelFunction','rbf');
end
```

softsvm:

```
%% SOFT Support Vector Machine
%%  Learns an approximately separating hyperplane for the provided data.
%% Inputs -
%%  X - Matrix with observations in each row.
%%  t - Vector of length equal to the number of columns of X, with either a 1 or -1
%%     indicating the class label.
%%  gamma - Slack penalty parameter. Higher implies greater violation penalty.
%% Outputs -
%%  w - Normal vector for the output hyperplane (plane equation is <w,x> + b = 0)
%%  b - Constant offset for the output hyperplane.

function [w, b,slack] = softsvm(X, t, gamma)
[N,D]=size(X);
```

```
%Build H- confident is good
H=zeros(N+D+1, N+D+1);
I=eye(D);
H(N+1:N+D,N+1:N+D)=I;

% Build f -confident
f=zeros(N+D+1,1);
f(1:N)=gamma;

%Build A (-In -TX -t)
A=zeros(N,N+D+1);
A(:,N+D+1)=-1*t;
A(:,1:N)=-1*eye(N);
T=diag(t);
A(:, N+1:N+D)=-T*X;

% A*x < bupper
b=zeros(N,1);
b(1:N)=-1;

%build lowerbound
lb=zeros(N+D+1,1);
lb(1:N)=0;
lb(N+1 : N+D+1)=-inf;
% structure is min 1/2(x'Ax) + f'x
%x=(psi omega b )
% psi is lenght N, slack
% w is length D, b is is sclar -> x is lenght D+N+1
x=quadprog(H,f,A,b,[],[],lb);
w=x(N+1: N+D);
b=x(N+D+1);
slack=x(1:N);
end
```

<u>evaluate_features_naivesvm</u>

```
%This function evaluates test data using the SVM from lab 7.
function [ L ] = evaluate_features_naivesvm(X_test, number_of_samples,
number_of_features_per_sample,w,b )
%takes matrix of X=(N*J)x42 (42 is length of MPCC) and
% w is
%returns a vector of labels L
%each song has J MPCC
% we apply the SVM (i.e. w,b ) to each feature (we get multiple features from a song)
% then we sum the results and decide by a majority vote
```

```
L=zeros(number_of_samples,1);
for i=1:number_of_samples
        feature_eval=X(((i-1)*number_of_features_per_sample+1):(i*number_of_feature
s_per_sample),:)*w +b;
        L(i)=sum(feature_eval);
end
end
```

<u>evaluate_features_builtinsvm:</u>

```
%This function evaluates the test data using the built in SVM model
function [ L ] = evaluate_features_builtinsvm(X_test, number_of_samples,
number_of_features_per_sample,svmmodel )
%takes matrix of X=(N*J)x42 (42 is length of MPCC) and
% w is
%returns a vector of labels L
%each song has J MPCC
% we apply the SVM (i.e. w,b ) to each feature (we get multiple features from a song)
% then we sum the results and decide by a majority vote.
L=zeros(number_of_samples,1);
for i=1:number_of_samples
[label,~]=predict(svmmodel,X(((i-1)*number_of_features_per_sample+1):(i*number_of_
features_per_sample),:));
   L(i)=sum(label);
end
end
```