

A Three-Feature Speech/Music Classification System

Courtenay Cotton

December 14, 2006

**Final Project
ELEN E4810
Digital Signal Processing
Prof. Ellis**

1. Introduction

The problem of automatically separating music signals from speech signals has been extensively studied. In general, approaches to this problem consider a small set of features to be extracted from the input signals. These features are carefully chosen to emphasize signal characteristics that differ between speech and music. This project combined two well-established features used to distinguish speech and music, as well as a third more novel feature. Once the typical values of these features were defined by a set of training data, a decision system for classifying future samples was chosen. Here, a simple k - nearest neighbor algorithm was implemented to determine whether an incoming sample should be considered speech or music. The implementation considered here treats each sample as a whole and labels the entire sample as either speech or music.

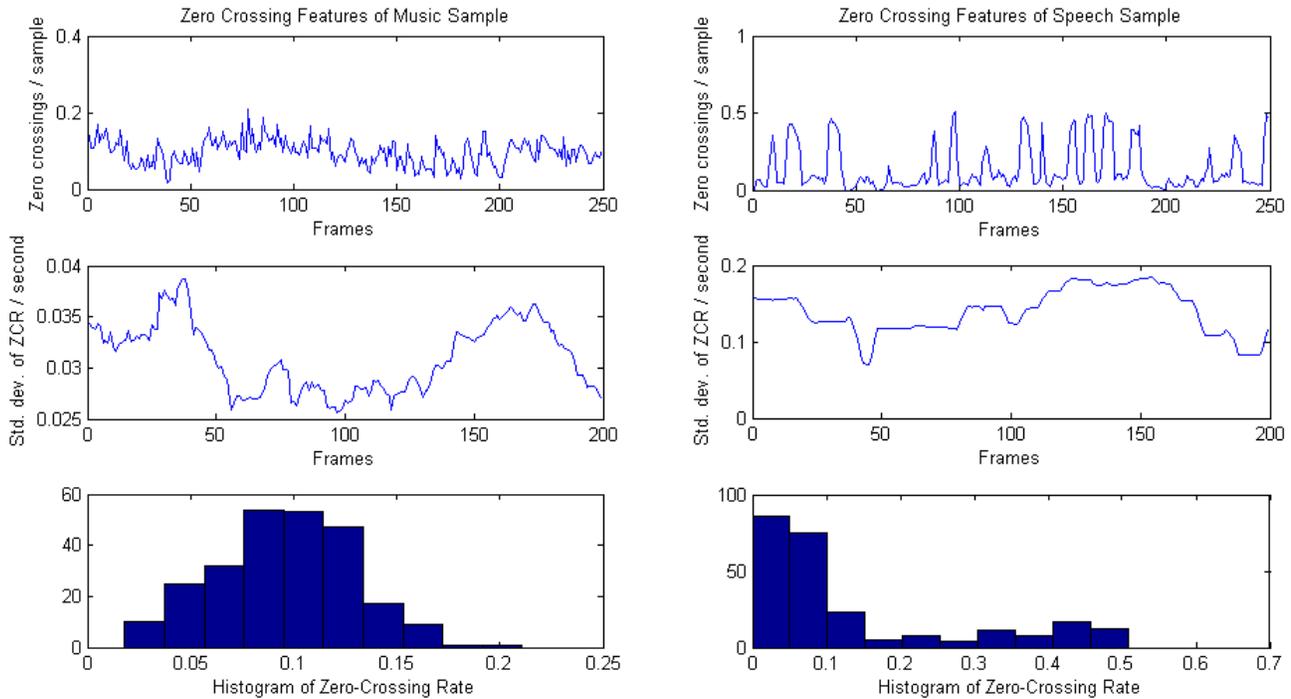
2. Signal Features

A large number of signal features have been employed for the problem of discriminating speech and music [1]. This project used two well-established features, the zero-crossing rate (specifically the variance of this rate) and the percentage of low-energy periods relative to the RMS value of the signal. The third feature used was a novel measurement of the residual error signal produced by linear predictive coding. These three features were combined to improve the robustness of the classification system and to hopefully balance out any ambiguities in any single feature set.

2.1. Variance of Zero-Crossing Rate

For this feature, the number of time-domain zero-crossings in each frame of the signal is calculated. While this rate is somewhat useful in distinguishing speech and music, a more useful measure is the short-term variance in the zero-crossing rate [2]. Figure 1 shows the zero-crossing features for a typical 5-second sample of music. Figure 2 shows the equivalent for a sample of speech. While the mean zero-crossing rate is approximately equal for both, the distribution of zero-crossings is very different. Music has a fairly normal distribution of frames with lower and higher zero-crossing rates. Speech however displays a much more skewed distribution. It demonstrates long periods with a low zero-crossing rate and very distinct periods with a smooth transition to a much higher zero-crossing rate. As a result of these characteristics, the average variance in the zero-crossing rate of speech signals tends to be higher than that of music signals.

In this implementation, the zero-crossing rate (number of zero-crossings per sample) was calculated for each 20 ms frame of a sample's data. Then the local variance of the zero-crossing rate was calculated over each second of data (with 50 frames of data per second). Finally the mean of the local variances was taken to be that sample's data value for the zero-crossing variance feature.

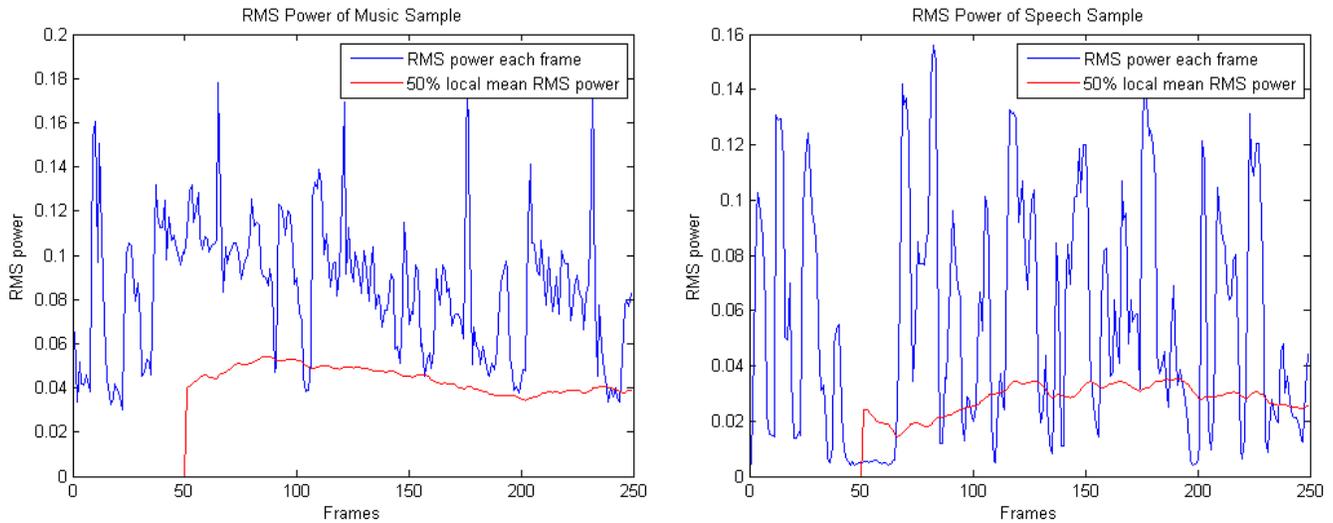


Figures 1 and 2. Zero-crossing features of a typical music sample and a typical speech sample. Top is the rate of zero-crossings for each frame. Middle is the standard deviation of the zero-crossing rate as calculated over all frames within the last second of data. Bottom is a histogram of the zero-crossing rate.

2.2. Percentage of Low-Energy Frames

This feature is based on the fact that music tends to be more continuous than speech and therefore has fewer instances of quiet frames. To quantify this, each frame is compared with the local frames' average root mean square (RMS) power. Quiet frames will tend to fall below 50% of this average RMS value [1]. Therefore the percentage of frames with power less than 50% of the local (within a second) average is a good distinguisher between music and speech signals. This percentage should generally be higher for speech than music. Figures 3 and 4 show the RMS power in each frame of data for a typical music and speech signal. They also show the threshold 50% below the local mean RMS power, used to identify "low-energy frames".

In this implementation, the RMS power in each 20 ms frame was calculated. Then the local mean of the RMS power was calculated over the last second of data. For each frame after the initial second of data, the frame was counted as a low-energy frame if it fell below 50% of this local mean. The number of low-energy frames in a sample over the total number of frames was taken as the sample's data value for this feature.



Figures 3 and 4. RMS power per frame in a typical music and speech sample. The red line is 50% of the mean of the local RMS power (where local is considered to be within the last second of data). Frames with RMS power below this line are considered “low-energy frames”.

2.3. Linear Predictive Coding Residual

Linear predictive coding (LPC) is a technique used to encode (typically) speech data so that less data can be used to transmit the same basic speech signal. LPC is based on a simple model of speech. It assumes the human vocal tract can be modeled as an all-pole filter through which an impulse train and white noise can be passed to create the sounds of speech. Figure 5 shows this model. During speech, periods of voiced speech (vocal cord vibrations) are modeled as an impulse train, while periods of unvoiced speech (fricatives and plosives) are considered to be equivalent to white noise, and therefore do not add information to the signal. Consequently the signal can be transmitted as merely

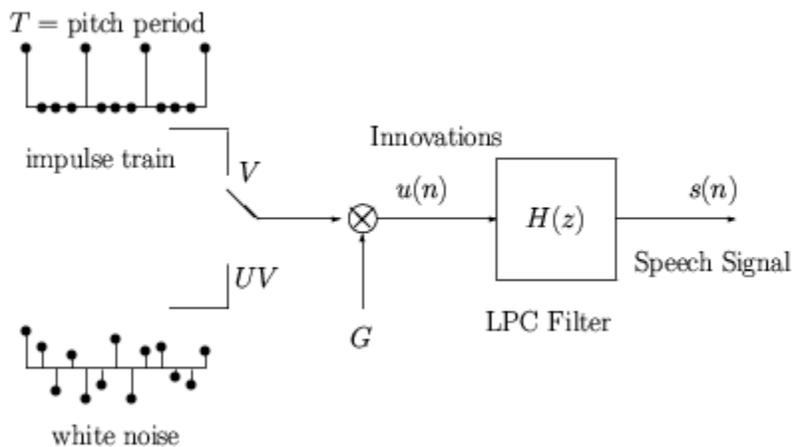
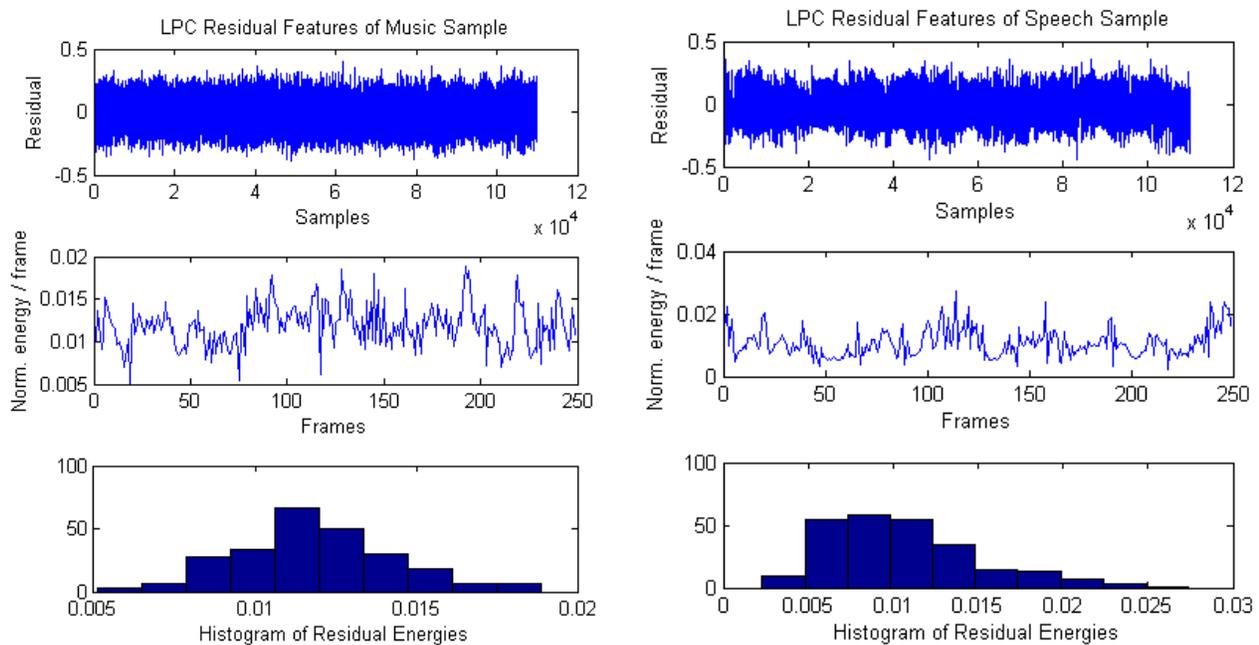


Figure 5. Linear predictive coding model of speech

the coefficients of the LPC filter and the impulse train, which is less data than the original signal. LPC analysis to find appropriate filter coefficients for each segment of a signal is typically performed on 20-30 ms overlapping frames of a signal. Conversely, the original signal can be passed through the inverse of the calculated LPC filters, to produce what is referred to as the residual [3].

The coefficients of the LPC filters of a signal have sometimes been used as a feature in discriminating speech from music. A somewhat more novel approach is to consider the residual signal itself and its properties as a feature of speech signals [4]. In this project, the LPC residuals of several music and speech samples were examined to determine any usable patterns. Figures 6 and 7 show the residual signals obtained from typical samples of music and speech. They also show the normalized energy of the residual for each frame of the data and a histogram of the distribution of this energy.



Figures 6 and 7. LPC Residual features of a typical music and speech sample. Top is the residual signal itself. Middle is the normalized energy per frame. Bottom is a histogram showing the distribution of the residual energies per frame.

The initial observation that was made was that speech seems to have a much more variable residual than music, which typically seems fairly constant. Taking measures of the variance directly was not very successful, however. This led to calculation of the residuals' energy over each frame of data in an attempt to find a usable pattern. It was found that the distributions of residual energy were skewed in a noticeable way. Furthermore, while the residual energy of music seemed to vary around the mean randomly, the residual energy of speech seemed to have a lower mean and more peaks high above the mean than below it. This led to the conclusion that the number of large peaks in residual energy could be a useable feature to distinguish speech from music.

In this implementation, the LPC coefficients and gain were calculated using a MatLab implementation found in [5]. The calculations were made every 20 ms over 30 ms frames of data. Each frame of sample data was then run through the inverse of its LPC filter and divided by the gain. These frame-by-frame residuals were then scaled and added together to produce a single residual signal. The energy of each 20 ms frame was then calculated as the sum of the absolute values of the squared residuals and normalized by the number of samples in the frame. The percentage of frames with energy more than 25% above the mean was then taken as the data value of the sample for this feature.

3. Training Data

In order to obtain a set of baseline measurements of the above features for “typical” speech and music samples, a set of training data was used. The training data consisted of four samples each of pure music (no vocals) or pure speech, obtained from [6]. Each data sample was five seconds long and sampled at 22.05 kHz. Each of the three feature calculations was performed on all data samples, to obtain a set of values that could be considered typical of music and speech signals.

4. Classification System

Once a set of baseline feature measurements was established, a system was needed to classify future data samples as having features more like speech or music. A k-nearest neighbor algorithm was chosen, for simplicity. In this classification scheme, the n feature values of a sample are taken to represent a location in an n-dimensional feature space. Here the three features can be translated to a three-dimensional feature space. The distance between a new data point and every training data point is calculated and a vote is taken among the k closest training data points (k neighbors) to determine the classification of the new data sample. In this implementation, k was chosen as 3 because of the small size of the training data and also in order to prevent ties.

Also, the range covered by each of the three features was normalized to a unit length in order to weight each feature equally in the distance calculations. This ensured that larger-valued features did not dominate over smaller ones.

5. Test Data

Another set of data was used to determine the accuracy of the proposed features and classification system. This set of test data consisted of 23 samples each of music and speech. Each sample was between 4 and 7 seconds long, in order to keep it similar to the test data. The music samples were obtained from a large personal collection and attempts were made to include a wide variety of music types. The speech samples were obtained from a combination of [7] and [8] and attempts were made to include both male and female speakers and samples with both single and multiple voices.

6. Results

Using the k-nearest neighbor framework, the feature data was normalized and plotted in a unit square 3-dimensional space for easy visualization. Figure 8 shows one view of the feature space, with all training and test data displayed.

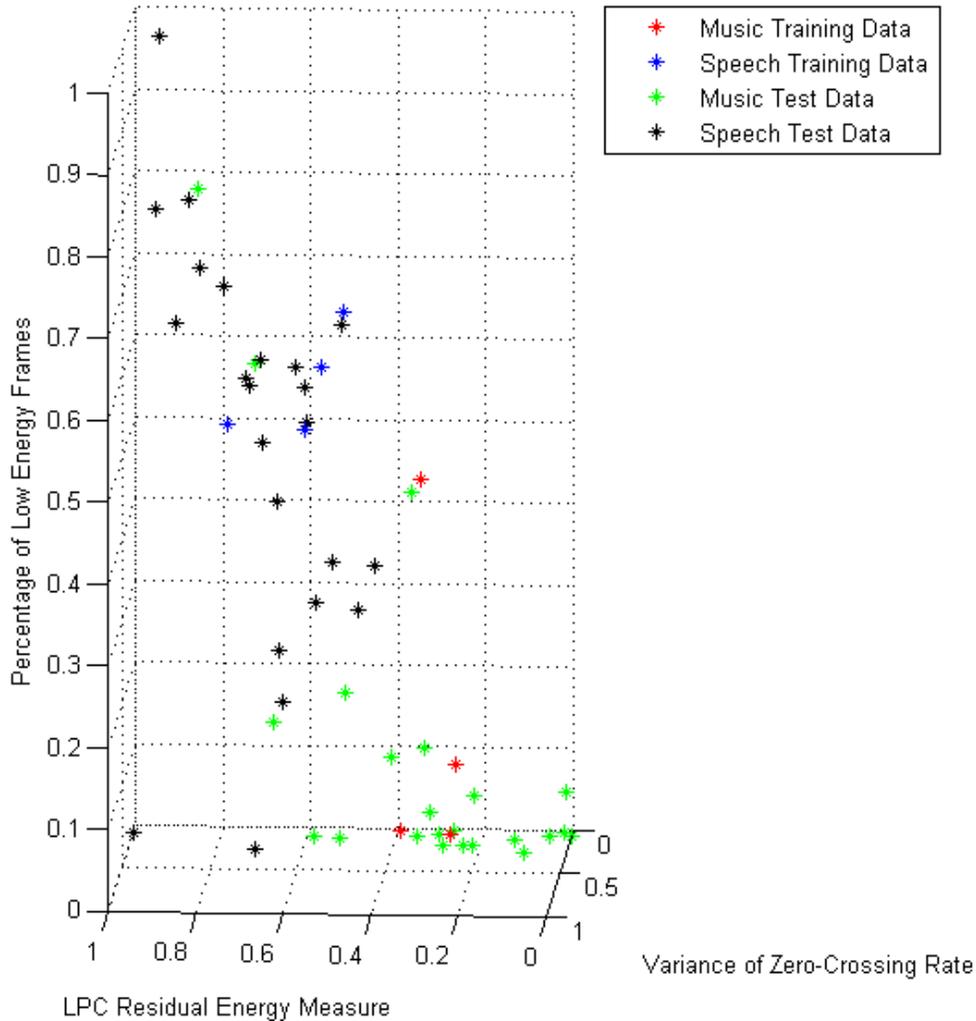


Figure 8. View of 3-dimensional feature space

As predicted, the speech samples displayed a higher percentage of low energy frames than the music samples. Similarly, the percentage of high peaks in the LPC residual energy was generally higher for speech and lower for music. Figure 9 shows another view of the feature space, with a clearer display of the zero-crossing data.

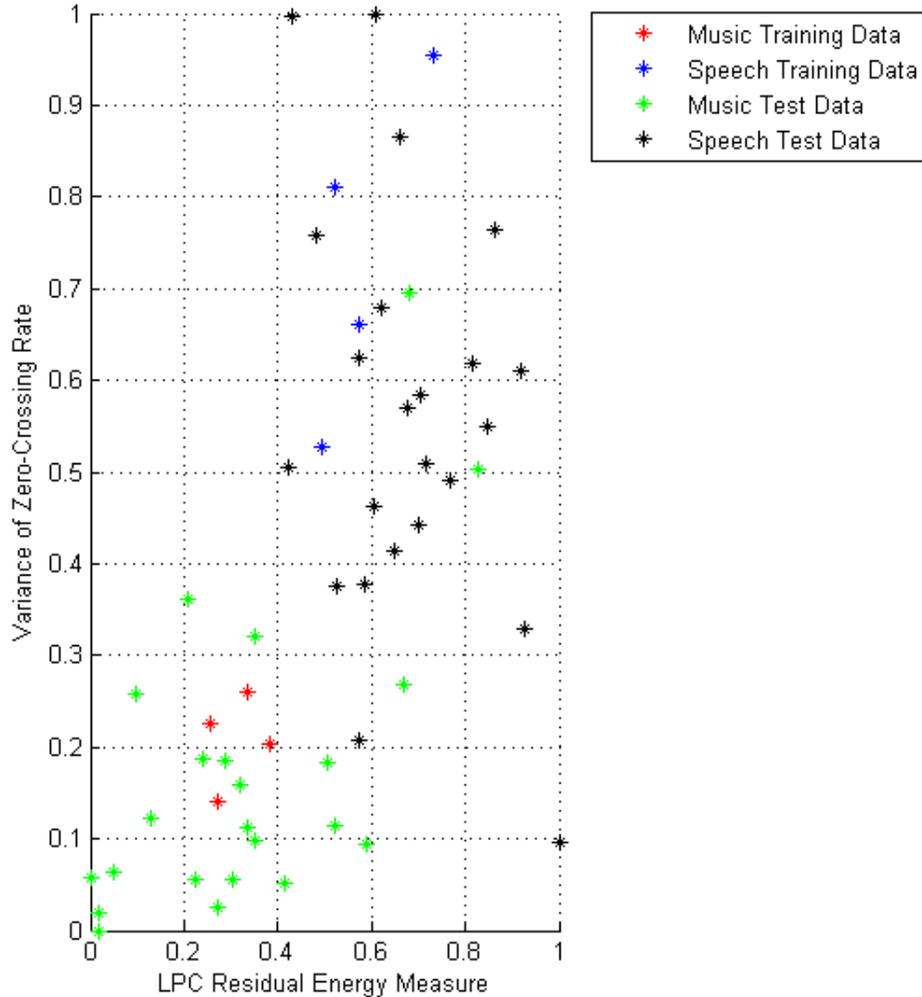


Figure 9. View of 2-dimensional space for zero-crossing and LPC residual features

Again, the samples behaved mostly as predicted, with speech samples tending to have a higher variance in zero-crossing rate. However, in both of these views it is clear that there were some outliers that did not follow the general pattern. Specifically, there are two music test samples that behaved like speech samples along all three of the feature axes. They would definitely be misclassified using this system. This may be because these two samples share some characteristics in common with speech and their differences from speech cannot be measured with the three features used here. When investigated, these two samples were both identified as having very dominant percussive sounds (possibly clapping in both cases) with little melody in between. It makes sense that these samples might be misclassified as speech since they are likely to have a high percentage of low energy frames and other speech-like qualities.

Another problem may be the one music training data sample that is very far removed from the others along the “low energy frames” axis. However, since it seems to cluster nicely with the music samples along the other two axes, it is assumed to be a usable training data point.

Using the k-nearest neighbor classification system described above, this system had the following success rates:

	Music Test Data	Speech Test Data	Total
Number Correct	17	22	39
Number Incorrect	6	1	7
% Correct	73.9%	95.7%	84.8%

7. Conclusions

This system proved to be fairly reliable. Although it was far more reliable for speech than for music, it still provided a high accuracy rate overall. A possible explanation for the disparity between speech and music classification accuracy may be that music as a concept comprises a much more varied set of signals than does speech. While the four samples of speech training data may have accurately defined the typical locations of speech samples in this feature space, the four samples of music may have been too few to accurately represent the locations of a variety of music. Another possible explanation may be that these three features are just not enough to reliably differentiate speech from most music. There may be an unexamined correlation between two or all three of the features used here, which would lower their combined usefulness (since the goal of using more than one feature is to add information to the system). Another failing of this implementation is that it only classifies entire samples of data as being speech or music. A more sophisticated discriminator would have the ability to determine the boundaries between speech and music in a continuous incoming signal, in real time. However, the same features used here could be implemented in such a system in order to partition an incoming data signal into periods of speech and/or music on a short-term basis.

8. References

- [1] E. Scheirer and S. Malcolm, "Construction and Evaluation of a Robust Multifeature Speech/Music Discriminator," *Proc. IEEE, ICASSP*, 1997.
- [2] C. Panagiotakis and G. Tziritas, "A Speech/Music Discriminator Based on RMS and Zero-Crossing," *IEEE Transactions on Multimedia*, Vol. 7, No. 1, pp. 155-166, February 2005.
- [3] J. Hai and E.M. Joo, "Improved Linear Predictive Coding Method for Speech Recognition," *ICICS-PCM 2003*, 3B3.2, pp. 1614-1618, 2003.
- [4] R.E. Yantorno, B.Y. Smolenski, and N. Chandra, "Usable Speech Measures and their Fusion," *ISCAS 2003*, Vol. III, pp. 734-737, 2003.
- [5] G.C. Orsak, et al, "Collaborative SP education using the Internet and MATLAB" *IEEE Signal Processing Magazine*, Vol. 12, No. 6, pp. 23-32, Nov. 1995.
- [6] D. Ellis. Music/Speech (Sound Examples), <http://www.ee.columbia.edu/~dpwe/sounds/musp/>, accessed Dec. 2006.

[7] WavList.com, <http://new.wavlist.com/>, accessed Dec. 2006.

[8] ReelWavs.com, <http://www.reelwavs.com/>, accessed Dec. 2006.

9. Appendix: Matlab Code (attached)

```
% Main - run all training and test data, plot results

%*****
% Training process:
%*****
% Set up training data (4 samples music / 4 samples speech):
train_music_files = {'music1.wav','music2.wav','music3.wav','music4.wav'};
train_speech_files = {'speech1.wav','speech2.wav','speech3.wav','speech4.wav'};

%*****
% Training: perform all 3 tests on training data

% Music samples:
for i = 1:length(train_music_files)
    train_lpc_music(i) = lpc_test(cell2mat(train_music_files(i)));
    train_zcr_music(i) = zcr_test(cell2mat(train_music_files(i)));
    train_rms_music(i) = rms_test(cell2mat(train_music_files(i)));
end
mean_train_lpc_music = mean(train_lpc_music);
mean_train_zcr_music = mean(train_zcr_music);
mean_train_rms_music = mean(train_rms_music);

% Speech samples:
for i = 1:length(train_speech_files)
    train_lpc_speech(i) = lpc_test(cell2mat(train_speech_files(i)));
    train_zcr_speech(i) = zcr_test(cell2mat(train_speech_files(i)));
    train_rms_speech(i) = rms_test(cell2mat(train_speech_files(i)));
end
mean_train_lpc_speech = mean(train_lpc_speech);
mean_train_zcr_speech = mean(train_zcr_speech);
mean_train_rms_speech = mean(train_rms_speech);

%*****
% Testing new data:
%*****
% Set up testing data:
music_files = dir(['m_*.wav']);
speech_files = dir(['s_*.wav']);

%*****
% Testing:

% Music samples:
for i = 1:length(music_files)
    lpc_music(i) = lpc_test(music_files(i).name);
    zcr_music(i) = zcr_test(music_files(i).name);
    rms_music(i) = rms_test(music_files(i).name);
end

% Speech samples:
for i = 1:length(speech_files)
    lpc_speech(i) = lpc_test(speech_files(i).name);
```

```
zcr_speech(i) = zcr_test(speech_files(i).name);
rms_speech(i) = rms_test(speech_files(i).name);
end

%*****
% Scale all three features between 0 and 1 to normalize distances:

min_lpc = min([min(train_lpc_music), min(train_lpc_speech), min(lpc_music), min
(lpc_speech)]);
max_lpc = max([max(train_lpc_music), max(train_lpc_speech), max(lpc_music), max
(lpc_speech)]);
min_zcr = min([min(train_zcr_music), min(train_zcr_speech), min(zcr_music), min
(zcr_speech)]);
max_zcr = max([max(train_zcr_music), max(train_zcr_speech), max(zcr_music), max
(zcr_speech)]);
min_rms = min([min(train_rms_music), min(train_rms_speech), min(rms_music), min
(rms_speech)]);
max_rms = max([max(train_rms_music), max(train_rms_speech), max(rms_music), max
(rms_speech)]);

train_lpc_music = (train_lpc_music - min_lpc)/(max_lpc - min_lpc);
train_zcr_music = (train_zcr_music - min_zcr)/(max_zcr - min_zcr);
train_rms_music = (train_rms_music - min_rms)/(max_rms - min_rms);
train_lpc_speech = (train_lpc_speech - min_lpc)/(max_lpc - min_lpc);
train_zcr_speech = (train_zcr_speech - min_zcr)/(max_zcr - min_zcr);
train_rms_speech = (train_rms_speech - min_rms)/(max_rms - min_rms);

lpc_music = (lpc_music - min_lpc)/(max_lpc - min_lpc);
zcr_music = (zcr_music - min_zcr)/(max_zcr - min_zcr);
rms_music = (rms_music - min_rms)/(max_rms - min_rms);
lpc_speech = (lpc_speech - min_lpc)/(max_lpc - min_lpc);
zcr_speech = (zcr_speech - min_zcr)/(max_zcr - min_zcr);
rms_speech = (rms_speech - min_rms)/(max_rms - min_rms);

%*****
% Combine independent measures into one matrix for each set of data:

music_training = [train_lpc_music; train_zcr_music; train_rms_music];
speech_training = [train_lpc_speech; train_zcr_speech; train_rms_speech];
music_testing = [lpc_music; zcr_music; rms_music];
speech_testing = [lpc_speech; zcr_speech; rms_speech];

%*****
% Plot 3-D decision space:

% Plot training data:
plot3(music_training(1,:), music_training(2,:), music_training(3,:), '*r');
hold on;
plot3(speech_training(1,:), speech_training(2,:), speech_training(3,:), '*b');

% Plot test data:
plot3(music_testing(1,:), music_testing(2,:), music_testing(3,:), '*g');
```

```

plot3(speech_testing(1,:), speech_testing(2,:), speech_testing(3,:), '*k');

% Label:
xlabel('LPC Residual Energy Measure');
ylabel('Variance of Zero-Crossing Rate');
zlabel('Percentage of Low Energy Frames');
legend('Music Training Data','Speech Training Data','Music Test Data','Speech Test Data');
grid on;

%*****
% K-Nearest Neighbor Classification: (K = 3) (0 = music, 1 = speech)

% Classify music samples:
music_classification = [];
for i = 1:length(music_testing)

    % Calculate distaces to each of 8 training samples
    for j = 1:length(music_training)
        dist(j) = sqrt((music_testing(1,i)-music_training(1,j))^2 + (music_testing(1,i)-
music_training(1,j))^2 + (music_testing(1,i)-music_training(1,j))^2);
    end
    for j = 1:length(speech_training)
        dist(j+4) = sqrt((music_testing(1,i)-speech_training(1,j))^2 + (music_testing(1,i)
-speech_training(1,j))^2 + (music_testing(1,i)-speech_training(1,j))^2);
    end

    % Find three closest neighbors
    sort_dist = sort(dist);
    n1 = find(dist == sort_dist(1));
    n2 = find(dist == sort_dist(2));
    n3 = find(dist == sort_dist(3));
    neighbor = [n1,n2,n3];
    neighbor = neighbor(1:3);

    % If 2 or more neighbors are music, classify as music
    % else classify as speech
    num_music_neighbors = find(neighbor <= 4);
    if (length(num_music_neighbors) >= 2)
        music_classification(i) = 0;
    else
        music_classification(i) = 1;
    end
end

% Classify speech samples:
speech_classification = [];
for i = 1:length(speech_testing)

    % Calculate distaces to each of 8 training samples
    for j = 1:length(music_training)
        dist(j) = sqrt((speech_testing(1,i)-music_training(1,j))^2 + (speech_testing(1,i)-
music_training(1,j))^2 + (speech_testing(1,i)-music_training(1,j))^2);

```

```
end
for j = 1:length(speech_training)
    dist(j+4) = sqrt((speech_testing(1,i)-speech_training(1,j))^2 + (speech_testing(1,
i)-speech_training(1,j))^2 + (speech_testing(1,i)-speech_training(1,j))^2);
end

% Find three closest neighbors
sort_dist = sort(dist);
n1 = find(dist == sort_dist(1));
n2 = find(dist == sort_dist(2));
n3 = find(dist == sort_dist(3));
neighbor = [n1,n2,n3];
neighbor = neighbor(1:3);

% If 2 or more neighbors are music, classify as music
% else classify as speech
num_music_neighbors = find(neighbor <= 4);
if (length(num_music_neighbors) >= 2)
    speech_classification(i) = 0;
else
    speech_classification(i) = 1;
end
end

%*****
% Calculate success measures:

music_correct = length(find(music_classification == 0));
speech_correct = length(find(speech_classification == 1));
total_correct = music_correct + speech_correct;

music_percent_correct = music_correct / length(music_classification);
speech_percent_correct = speech_correct / length(speech_classification);
total_percent_correct = total_correct / (length(music_classification)+length
(speech_classification));

disp(['Music Sample Classification: ', num2str(music_percent_correct*100), ' %']);
disp(['Speech Sample Classification: ', num2str(speech_percent_correct*100), ' %']);
disp(['Total Sample Classification: ', num2str(total_percent_correct*100), ' %']);

%*****
```

```
% lpc_test.m
% Perform linear predictive coding on sound sample and calculate percentage
% of high energy spikes in the residual signal

function [lpc] = lpc_test(wavfilename)

% Read wav file
[data, fs]= wavread(wavfilename);

% LPC Order
N = 10;

% Length of frame
frame_size = 0.020;
frame_length = round(fs*frame_size);
frames_per_sec = round(1/frame_size); % 50 frames per second

% Perform LPC analysis
[A,resid,stream] = lpcproc(data,fs,N);

% Find length of residual stream
len_samp = length(stream);

% Calculate normalized energy of each frame
energy = [];
n = 1;
for frame = 1:frame_length:len_samp-frame_length
    energy(n) = sum(abs(stream(frame:frame+frame_length-1)).^2)/frame_length;
    n = n + 1;
end
num_frames = length(energy);

% Calculate percentage of high spikes
highpoints = 0;
% If more than 1 second of frames
if (num_frames > frames_per_sec)
    for j = frames_per_sec+1:num_frames

        % Mean energy over last second
        meanEnergy(j) = mean(energy(j-frames_per_sec:j));

        if (energy(j) > 1.25*meanEnergy(j))
            highpoints = highpoints + 1;
        end
    end
end
lpc = highpoints/(num_frames-frames_per_sec);
```

```
% zcr_test.m
% Calculate zero-crossing rates for each frame of a sound sample and output
% the average one second variance of the zero-crossing rate

function [zcr] = zcr_test(wavfilename)

% Read wav file
[data, fs]= wavread(wavfilename);

% find length of wav file
len_samp = length(data);

% Length of frame
frame_size = 0.020;
frame_length = round(fs*frame_size);
frames_per_sec = round(1/frame_size); % 50 frames per second

% Calculate number of zero-crossings in each frame
zcr = [];
n = 1;
for frame = 1:frame_length:len_samp-frame_length
    frameData = data(frame:frame+frame_length-1);

    % Sum up zero crossings accross frame
    zcr(n) = 0;
    for i = 2:length(frameData)
        zcr(n) = zcr(n) + abs(sign(frameData(i)) - sign(frameData(i-1)));
    end
    zcr(n) = zcr(n)/(2*frame_length);

    n = n + 1;
end
num_frames = length(zcr);

% Calculate variance in zero-crossing rate from last second of data
lef = 0;
% If more than 1 second of frames
if (num_frames > frames_per_sec)
    k = 1;
    for j = frames_per_sec+1:num_frames
        std_zcr(k) = std(zcr(j-frames_per_sec:j));
        k = k + 1;
    end
end

% Result is mean 1-sec variance in zero crossing rate
zcr = mean(std_zcr);
```

```
% rms_test.m
% Calculate the RMS values over frames of a sound sample and return the
% percentage of low energy frames in the sample

function [rms] = rms_test(wavfilename)

% Read wav file
[data, fs]= wavread(wavfilename);

% find length of wav file
len_samp = length(data);

% Length of frames
frame_size = 0.020;
frame_length = round(fs*frame_size);
frames_per_sec = round(1/frame_size); % 50 frames per second

% Calculate RMS value of each frame
rms = [];
n = 1;
for frame = 1:frame_length:len_samp-frame_length
    frameData = data(frame:frame+frame_length-1);

    % Calculate RMS value of frame
    rms(n) = sqrt(sum(frameData.^2)/length(frameData));
    n = n + 1;
end
num_frames = length(rms);

% Calculate number of low energy frames
lef = 0;
% If more than 1 second of frames
if (num_frames > frames_per_sec)
    for j = frames_per_sec+1:num_frames
        meanRMS(j) = mean(rms(j-frames_per_sec:j));
        if (rms(j) < 0.5*meanRMS(j))
            lef = lef + 1;
        end
    end
end

% Result is percentage of low energy frames
rms = lef/(num_frames-frames_per_sec);
```

```
% lpcproc.m
% Perform LPC analysis on input data
% Return LPC filter coefficients, residuals, and residual stream
% Adapted from Reference [5]:
% G.C. Orsak, et al, "Collaborative SP education using the Internet and
% MATLAB" IEEE Signal Processing Magazine, Vol. 12, No. 6, pp. 23-32, Nov. 1995.
```

```
function [A,resid,stream] = lpcproc(data,fs,N,frameRate,frameSize)
```

```
if (nargin<3), N = 13; end
if (nargin<4), frameRate = 20; end
if (nargin<5), frameSize = 30; end
```

```
preemp = .9378;
```

```
[row col] = size(data);
if col==1 data=data'; end
```

```
% Set up
nframe = 0;
samp_between_frames = round(fs/1000*frameRate);
samp_per_frame = round(fs/1000*frameSize);
duration = length(data);
samp_overlap = samp_per_frame - samp_between_frames;
```

```
% Function to add overlapping frames back together
ramp = [0:1/(samp_overlap-1):1]';
```

```
% Preemphasize speech
speech = filter([1 -preemp], 1, data)';
```

```
% For each frame of data
for frameIndex=1:samp_between_frames:duration-samp_per_frame+1
```

```
    % Pick out frame data
    frameData = speech(frameIndex:(frameIndex+samp_per_frame-1));
    nframe = nframe+1;
```

```
    autoCor = xcorr(frameData); % Compute the cross correlation
    autoCorVec = autoCor(samp_per_frame+[0:N]);
```

```
    % Levinson's method
    err(1) = autoCorVec(1);
    k(1) = 0;
    a = [];
    for index=1:N
        numerator = [1 a.']*autoCorVec(index+1:-1:2);
        denominator = -1*err(index);
        k(index) = numerator/denominator; % PARCOR coeffs
        a = [a+k(index)*flipud(a); k(index)];
        err(index+1) = (1-k(index)^2)*err(index);
    end
```

```
% LPC coefficients and gain
A(:,nframe) = [1; a];
G(nframe) = sqrt(err(N+1));

% Inverse filter to get error signal
errSig = filter([1 a'],1,frameData);
resid(:,nframe) = errSig/G(nframe);

% Add residuals together by frame to get continuous residual signal
if(frameIndex==1)
    stream = resid(1:samp_between_frames,nframe);
else
    stream = [stream;
    overlap+resid(1:samp_overlap,nframe).*ramp;
    resid(samp_overlap+1:samp_between_frames,nframe)];
end
if(frameIndex+samp_between_frames+samp_per_frame-1 > duration)
    stream = [stream; resid(samp_between_frames+1:samp_per_frame,nframe)];
else
    overlap = resid(samp_between_frames+1:samp_per_frame,nframe).*flipud(ramp);
end
end

stream = filter(1, [1 -preemp], stream)';
```