EE513 Final Project: Three State Classifier

Matt Ruffner
matthew.ruffner@uky.edu

ABSTRACT

In this project, linear and quadratic minimum distance classifiers were implemented with MATLAB in order to discern 3 different categories of recorded speech. The quality of Features extracted from the dataset was verified using Fisher's criterion in order to determine how statistically independent they were. Decision thresholds were employed for actually determining which category a certain audio clip was part of.

1. APPROACH

The audio segment was first trimmed to avoid recorded noise other than that of the Speaker of Interest. It was then split in half and the Mel Cepstrum values were computed using the provided solution to Quiz 5.

Among the features chosen to construct the classifier, the 1st, 3rd and 8th delta value taken from the cepstrum were used. In addition, the ratios of formant frequencies 3 and 2, as well as 4 and 2. The first formant was measured in the first half of the speech segment, the second in the latter half.

After analyzing all of the training data at once, the Fisher criterion of the selected features is shown in Fig. **??**.

No bootstrapping was preformed since the Fishers criterion for each feature was deemed to be high enough.

Thresholding was done by analyzing the average distance metrics as well as the average of their differences. These were used to analytically derive a decision threshold.

The scaled covariance matrix as well as the two class template mean vectors are stored in `cov.mat` to be read in by the testing script `runme.m`

2. RESULTS

The recall and precision for 'yes' and 'no' scenarios are acceptable. A more exhaustive approach to finding threshold criteria for classifying the 'neither' would have proved worthwhile.

ACM ISBN N.A..
DOI: N.A.

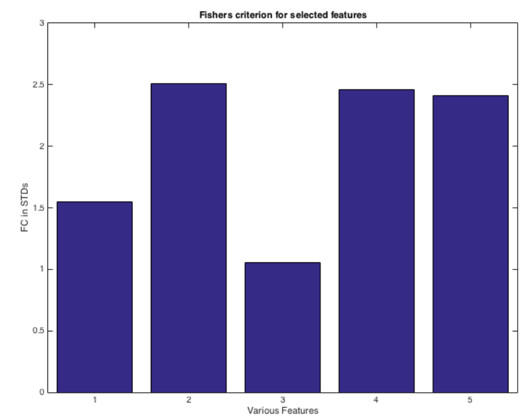

Figure 1: Fishers criterion for each of the selected features averaged over the entire dataset, shown in terms of standard deviations.

	YES	NO	NEITHER	Precision
YES	42	1	15	72%
NO	0	48	12	80%
NEITHER	13	6	28	60%
Recall	76%	87 %	51%	

Table 1: The resulting confusion matrix after classifying test data with the final covariance matrix generated.

3. CONCLUSIONS

In hindsight, bootstrapping in order to find more optimum combinations of features would have been a beneficial practice. However, delta values from the cepstrum prove to have a high uniqueness which works quite well.

More work could be put into making sure classifications are made with higher precision for 'NEITHER' cases, since in a practical application a misclassification here might be unacceptable.

APPENDIX

```
% EE513 Final Project
% Matt Ruffner
% 5/2/17
%
% script to test and train on yes/no data
```

```matlab
% to determine an optimal classifier
% much code adapted from Dr. Kevin Donohue's mindistex.m, featureexhw.m and the provided solution to HW7.

% specify wav file parent directory here
audiopath = 'YesNoFinalProject/';

% creates cell arrays of each file type
yfiles = dir(strcat(audiopath,'y*.wav')); yfiles={yfiles.name};
nfiles = dir(strcat(audiopath,'n*.wav')); nfiles={nfiles.name};
afiles = {yfiles nfiles}; % all files to index into

kboot = 1;  % number of times to bootstrap
bsize = 55; % number of files in each bootstrap

% feature vector
feat = {};
% feat{1} —> 'yes' recordings
% feat{2} —> 'no' recordings


% feature vector indexes for yes/no
for type=1:2
    dels{type}=[];

    % create random sample of recordings
    files = randsample(afiles{type},bsize);

    % iterate over random sample of recordings
    fcount=1;
    for rec=files

        [y,fs] = audioread(char(strcat(audiopath,rec))); % read in audio
        [b,a] = butter(4, 2*[200]/fs,'high'); % room noise filter
        yf = filtfilt(b,a,y); % apply room noise filter
        [ytrim,kb,ke] = trimit(yf,fs);
        mid = floor(length(ytrim)/2);
        bytrim = ytrim(1:mid);  % Get first half data
        eytrim = ytrim(mid+1:end); % Get last half data

        del = mfcc(bytrim,fs) — mfcc(eytrim,fs);


        [lpsees1,er] = lpc(bytrim,12);  %  LPC coefficents with model order 12
        [lpsees2,er] = lpc(eytrim,12);  %  LPC coefficents with model order 12
        r1 = roots(lpsees1);   %  Find poles of LPC reconstruction filter
        r2 = roots(lpsees2);   %  Find poles of LPC reconstruction filter
        sf1 = fs*angle(r1)/(2*pi);  %  Find angles corresponding to poles
        sf2 = fs*angle(r2)/(2*pi);  %  Find angles corresponding to poles
        %  Find those corresponding to complex conjugate poles
        %  and in an expcted range of frequencies
        nf = find(sf1 > 250 & sf1 < (fs/2 — 100));
        r1 = r1(nf);  %  Trim to get relevant/positive roots
        sf1 = sf1(nf);  % Corresponding frequenies
        ff1 = sort(sf1);  %  Order frequencies from smallest to largest
        nf = find(sf2 > 250 & sf2 < (fs/2 — 100));
        r2 = r2(nf);  %  Trim to get relevant/positive roots
        sf2 = sf2(nf);  % Corresponding frequenies
        ff2 = sort(sf2);  %  Order frequencies from smallest to largest

        % save features
        feat{type}(1,fcount) = del(1);
        feat{type}(2,fcount) = del(3);
        feat{type}(3,fcount) = del(8);


        % third formant of first half to 2nd of second half
        feat{type}(4,fcount) = ff1(3)/ff2(2);

        feat{type}(5,fcount) = ff1(4)/ff2(2);

        fcount=fcount+1;
    end
end


%  Build classifier without scaling
mu1 = mean(feat{1},2);  %  Template for class 1
mu2 = mean(feat{2},2);  %  Template for class 2

%  Apply minimum distance classifier with no scaling to class 1 samples
%  (Training set)
d11 = sqrt(sum((feat{1}—mu1*ones(1,bsize)).^2));  % compare class 1 samples with template 1
d12 = sqrt(sum((feat{1}—mu2*ones(1,bsize)).^2));  % compare class 1 samples with template 2
%  Find all the incorrect classification (note all should be closest to
%  class 1 templte for this case)
cd1 = find(d11 > d12);  %  Length of cd1 is the number of mis—classifications

%  Apply minimum distance classifier with no scaling to class 2 samples
%  (Training set)
d21 = sqrt(sum((feat{2}—mu1*ones(1,bsize)).^2));
d22 = sqrt(sum((feat{2}—mu2*ones(1,bsize)).^2));
%  Find all the incorrect classification (note all should be closest to
%  class 2 templte for this case)
cd2 = find(d22 > d21); %  Length of cd1 is the number of mis—classifications
%  Overall classification error with scaling
classerr = 100*(length(cd1)+length(cd2))/(2*bsize)
```

```matlab
%  scale distances by covariance matrix

%  Compute covariance matrix for each class and assume variations
%  over each class is statistically the same.  Note since means
%  are different in each class so these must be removed within each class
%  before computing the covariance matrix.  This can be done
%  with matlab COV command applied to each class as shown below:

covtot = (cov(feat{1}')+cov(feat{2}'))/2;  % Average covariances from each class together
cinv = inv(covtot);  % Take inverse


%  Apply minimum scaled distance classifier with scaling to class 1 samples
%  (Training set)
d11ws = zeros(1,bsize);  % Initalize distance vectors with zeros since we index these in a loop
d12ws = zeros(1,bsize);
d21ws = zeros(1,bsize);
d22ws = zeros(1,bsize);
%  Apply minimum distance classifier with scaling to class 2 samples
%  (Training set)

%  Class 1 distances to each template
for k=1:bsize
    d11ws(k) = (feat{1}(:,k)-mu1)'*cinv*(feat{1}(:,k)-mu1);  % compare class 1 samples with template 1
    d12ws(k) = (feat{1}(:,k)-mu2)'*cinv*(feat{1}(:,k)-mu2);  % compare class 1 samples with template 2
end
%  Find all the incorrect classifications (note all should be closest to
%  class 1 templte for this case)
cd1s = find(d11ws > d12ws);  %  Length of cd1s is the number of mis-classifications

% Class 2 distances to each template
for k=1:bsize
    d21ws(k) = (feat{2}(:,k)-mu1)'*cinv*(feat{2}(:,k)-mu1);  % compare class 2 samples with template 1
    d22ws(k) = (feat{2}(:,k)-mu2)'*cinv*(feat{2}(:,k)-mu2);  % compare class 2 samples with template 2
end
%  Find all the incorrect classifications (note all should be closest to
%  class 2 templte for this case)
cd2s = find(d22ws > d21ws); %  Length of cd2s is the number of mis-classifications
%  Overall classification error with scaling
classerrws = 100*(length(cd1s)+length(cd2s))/(2*bsize)

figure(1)
fc = abs(mean(feat{2}') - mean(feat{1}')) ./ sqrt ((std(feat{2}').^2 + std(feat{1}').^2)/2);
bar(fc)
xlabel('Various Features')
ylabel('FC in STDs')
title('Fishers criterion for selected features')


save('cov', 'cinv', 'mu1', 'mu2')
```