Signal och system Projekt ht 02

Röstigenkänning

av

Fabyanna Knutar	770212-4525
Teb Björling	800216-0052
Magnus Sundin	720409-1412
Peter Sundin	780319-0490

Innehåll

Sammanfattning 3
Introduktion 3
Lösning 3
  Inläsning av ljudfiler 4
  Trunkering av ljudfiler 4
  Samples in i block 4
  Fönstrar varje block 5
  FFT 5
  Mel-frekvens 6
  Cepstrum 6
  Klustring 7
  Jämförelse av olika röster 8
Resultat 9
  Problem och möjliga förbättringar 9
Slutsatser 10
Appendix 11
  Källkoder 11
Referenser 20

Sammanfattning

Att konstruera ett system som automatiskt kan detektera vem som pratar, baserat på en referensdatabas. En applikation som har många användningsområden, bland annat många kommersiella.
Den metod vi använder visar sig fungera mycket bra, och redan våra första tester visade ett 100%-igt korrekt resultat, även då våra testfiler varit slumpartade och haft en kort längd.

Introduktion

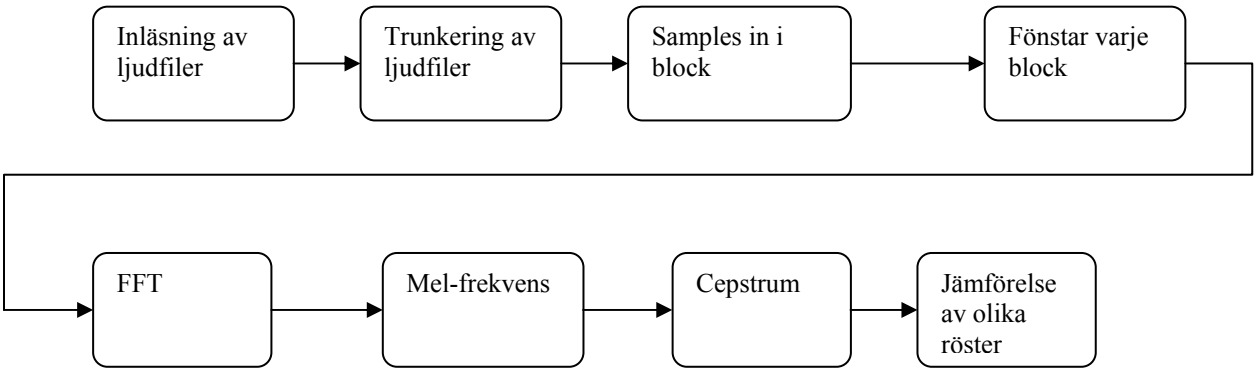
Varför är det intressant med röstigenkänning? Om man kan känna igen röster automatiskt, så finns det många tänkbara användningsområden. Till exempel vid inloggningsförfaranden, automatisk inladdning av personliga inställningar t.ex. i bilen m.m. Det är fantasin som sätter gränserna, och överallt där man kan tänka sig att snabb åtkomst av personlig data är av vikt så är röstigenkänning en teknik att överväga.

Varför kan man känna igen röster? Lösningen ligger i att vi alla har ett relativt unik röstspektrum i frekvensdomänen, vilket gör att extraktion av den mest signifikanta av denna kan användas då man vill skapa ett ”fingeravtryck”. Den största svårigheten ligger just i framtagandet av den viktiga informationen som våra röster består av, dels för att en för applikationen anpassad dimensionsreducering krävs, dels för att kunna tolka resultatet av denna korrekt. Tillvägagångssätten för att lösa detta problem är många till antal, och fokuserar på olika användningsområden. Den metod vi valde, efter några initialtester, baseras på olinjär frekvenstransformation enligt Mel-frekvenser och klustring genom Vektor Kvantiserings efter den s.k. LBG-algoritmen.

Lösning

Vår första idé var att göra en FFT på en längre inspelad röst. Och sedan dela upp frekvensspektrat I standardiserade block. Ta medelvärdet inom varje block och spara det som en profil. Men innan vi kom så långt att vi kunde implementera och testa detta, så fann vi på en annan metod på nätet. En metod som verkade vara mer avancerad och empiriskt förankrad samtidigt som den enligt utsagor skulle fungera väldigt bra. Det som gjorde den extra attraktiv var att den är indelad i ett flertal konkreta steg, som alla verkade vara väl dokumenterade och verifierade. Vår approach blev således att metodiskt gå igenom dessa nivåer, för att så att säga under resans gång kunna vara försäkrade om att det vi hittills implementerat fungerar.

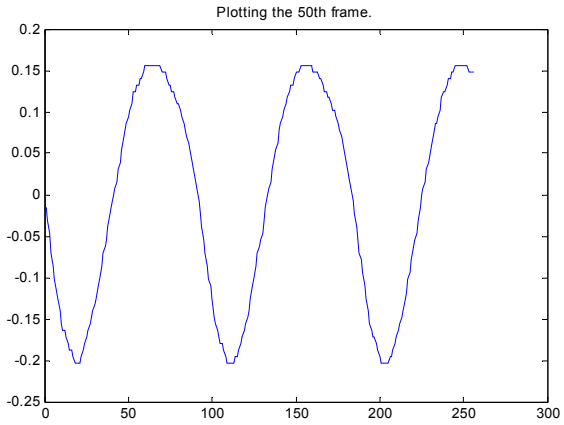
I princip går vår lösning ut på att genom Matlab känna igen tidigare inspelade röster sparade i en databas i .wav format.



Inläsning av ljudfiler
Vi läser in ett antal .wav filer från en databas. Dessa filer är naturligtvis inspelningar av röster, eftersom det ju är huvudmålet med detta projekt. Vi kan läsa in dessa i Matlab med hjälp av en funktion som heter wavread vilken returnerar signalen (rösten) i tidsdomänen. Vi får även tillbaka längden av signalen samt samplingsfrekvens.

Trunkering av ljudfiler
I detta steg klipper vi bort brus i början och i slutet av signalen. Detta brus uppstår och kan särskiljas genom att amplituden på signalen ligger under en bestämd gräns. Vi låter varje sample testas mot denna gräns för att se om den ligger under. Då en sample går över den av oss fördefinierade gränsvärdet klipper vi bort alla tidigare samples vilka redan jämförts med. Ett liknande förfarande sker för att klippa bort onödiga sample i slutet av ljudfilen. Detta gör i slutändan att vi får den signifikanta delen av signalen utan något inledande eller avslutande brus.

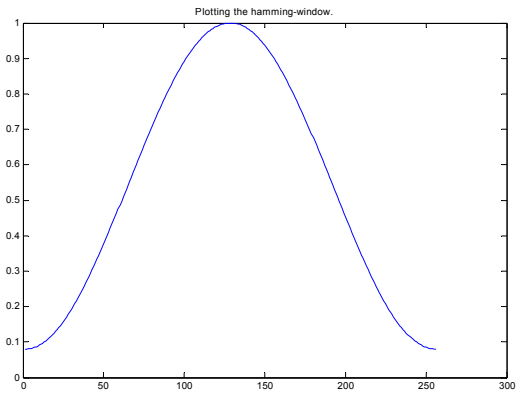
Samples in i block
I denna del fördelar vi N (vanligtvis och i detta fall 256) antal samples och stoppar in dem i delvis överlappande block. Den här processen fortsätter tills hela signalen blivit uppdelad på block. Anledningen till att blockstorleken vi använder är just 256 är att Fast Fourier-Transformen (FFT), som kräver blocklängder som är potenser av 2, blir mycket bekvämare att utföra samt att längden medger en utbredning i tidsdomänen för varje block av ungefär 20 millisekunder.
Tanken med att ha små block är att då får signalen utseende av endast mindre ändringar och blir näst intill stationär under dessa små tidsintervall, vilket gör signalen i stort lättare att analysera.

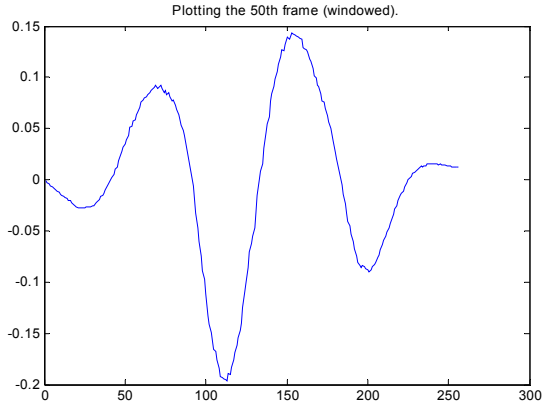


Fönstrar varje block

Här fönstrar vi varje enskild ram för att minimera signalernas diskontinuiteter i början och i slutet av varje ram. Grundidén är att minimera spektrats distortion genom att fönstra och på så sätt minska signalen till noll i början och i slutet av varje ram.

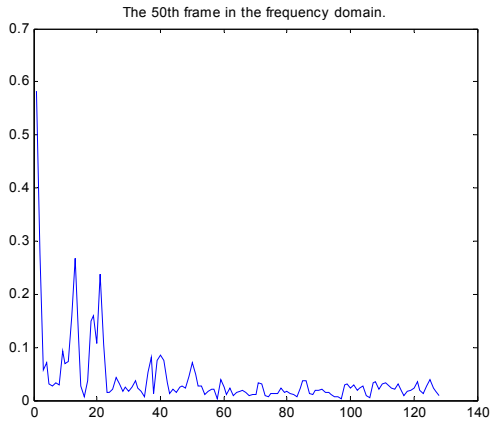
Vi använder oss av Hamming-fönster med blockstorleken som inparameter. Denna funktion fanns redan definierad i Matlab till vår stora glädje.





FFT

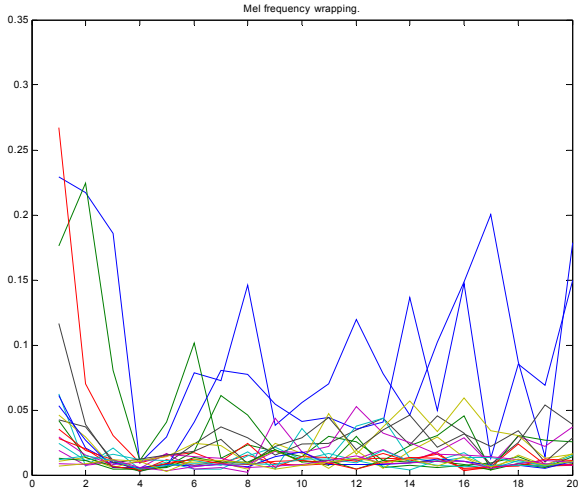
Nu måste vi göra en FFT av den aktuella signalen för varje block, för att föra in signalen från den otympliga tidsdomänen till den mer karaktäristiska frekvensdomänen. För att göra detta måste varje block vara en multipel av 2 och som tidigare påpekats delade vi in blocken i längder om 256 samples. Även denna funktion fanns fördefinierad i Matlab.



Mel-frekvens

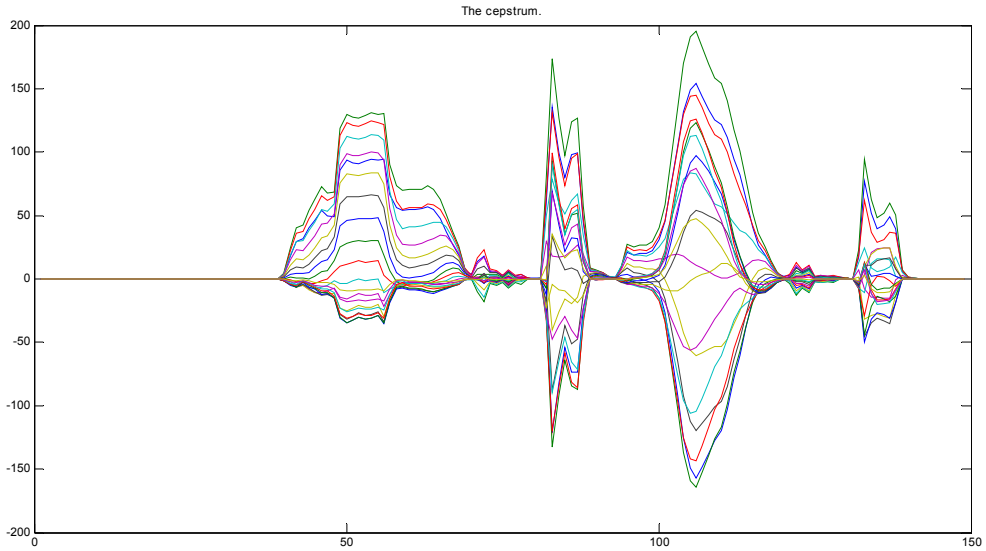
Att göra en Mel-frekvens-filterbank är en vedertagen metod vid röstigenkänning. Metoden implementerade vi som en funktion vilken anropas med tre inparametrar. Dessa tre parametrar är antal filter, längden av FFT:n från förra steget samt signalens samplingsfrekvens. Denna funktion retunerar en matris vilken innehåller en filterbank av signalens olika amplituder. Detta ger oss ett spektrum som består av lådor i frekvensdomänen.

Mel-frekvens, linjär förstärkning av frekvenser upp till 1kHz, de högre frekvenserna enl. algoritm...
Filterbanken är i princip flera bandpassfiler i rad. Vi valde att ha 20 stycken bandpass.



Cepstrum

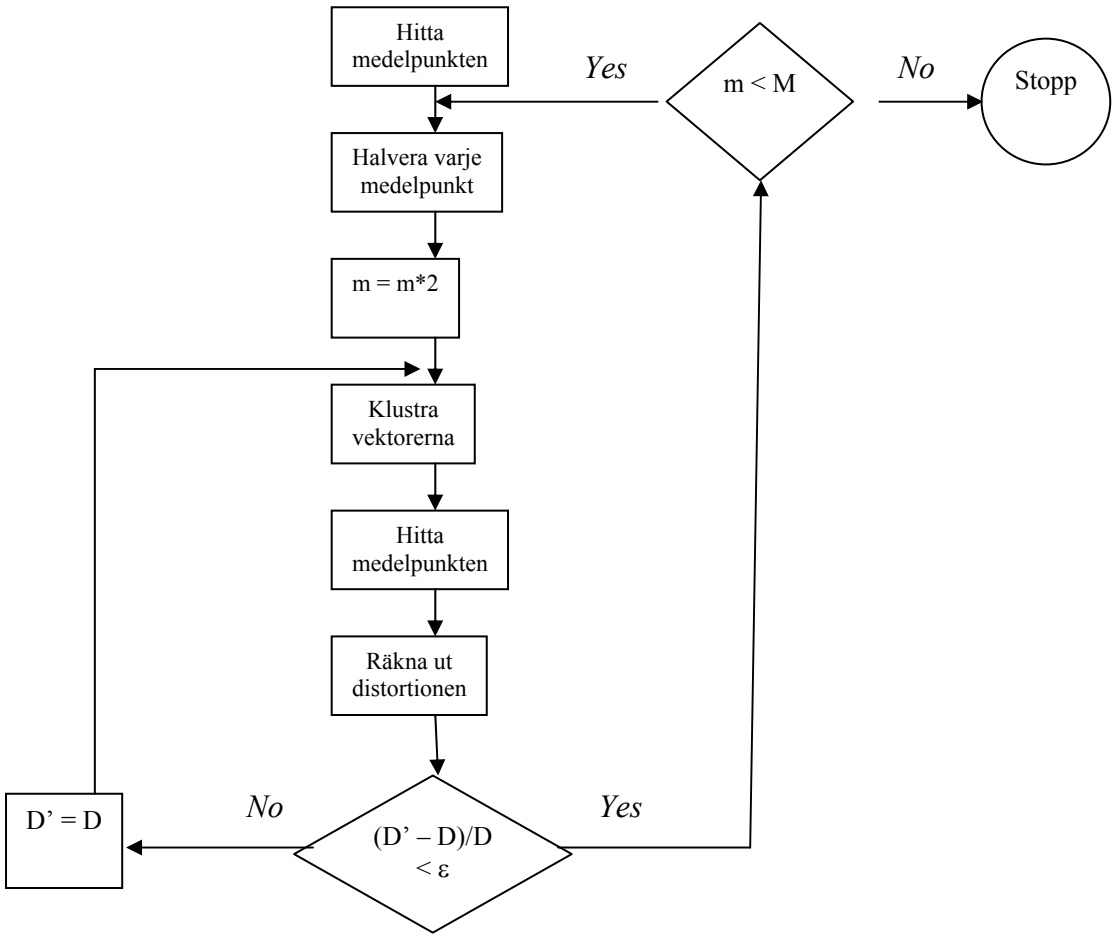
Detta slutgiltiga steg kommer vi att konvertera mels spektrum tillbaka till tidsdomänen. Resultatet kallas Mel Frequency Cepstrum Coefficients (MFCC). För att göra detta använder vi oss av en diskret kosinus transform. Denna funktion finns i Matlab.



Klustring
Klustring använder man när ett stort antal mätvärden skall kombineras till ett strikt färre antal karaktäristiska noder. Dessa kallas i detta sammanhang för centroider, och utgör medelvärdet för de mätvärden som tilldelats dem. Anledningen till att vi vill använda klustring är för att varje sampling består av så mycket information att en jämförelse mellan ljudfiler hade blivit svår och oerhört resurskrävande. Reduktion av noder är därför viktig för att göra problemet mer lätthanterligt.

Klustring i sig är ingen algoritm, utan det finns en uppsjö olika varianter, där den vi beslutade oss för att nyttja går under beteckningen Vector Quantization (VQ). Enklast möjliga vis att förklara hur denna metod fungerar är att tänka sig ett 2D-plan med ett antal mätpunkter. Första steget i processen är att initiera en begynnelsevektor som medelvärdet för mätmängden. Sedan delar man upp denna i två delar och ser vilka mätvärden som hör till vilken vektor (genom att se vilken centroid som genererar det minsta euklidiska avståndet till respektive mätpunkt). När denna indelning gjorts i dessa initialkluster beräknar man det nya medelvärdet för respektive klustret och flyttar centroiden dit. Nu gör man om tilldelningen, och flyttar centroiden till det nya medelvärdet. Detta förfarande fortgår tills centroiduppdateringen konvergerar. Då delar man åter upp varje centroid i två, och går igenom algoritmen. Detta fortsätter man med tills antalet önskade kluster i uppnått.

Vi beslutade oss för att följa Linde, Buzo, Gray – algoritmen (LBG), som är en känd algoritm för att implementera VQ:



Denna del av processen består av ett antal delsteg:

1. Räkna ut medelpunkten av cepstrumets delar och kallar den för centroid
2. Delar upp denna centrala punkt i två punkter genom att lägga till ett litet tal, epsilon. Detta skall göras tills medelvärdet har uppnåtts.
3. Nu ska dessa värden klustras kring dessa punkter. Detta görs genom att räkna ut längden från punkten till närliggande klusterpunker.
4. Genomför medelpunktsuträkningen igenom för att hitta de exakta värdena.
5. Genomför punkterna 2-4 igenom tills antal klusterpunkter har blivit genomförda.

Jämförelse av olika röster

Jämförelsen mellan vår referensdatabas och ”inputljudet” sker genom att jämföra avståndet mellan de olika klusterna. Den referensröst med det minsta sammanlagda avståndet till teströsten antas vara den riktiga.

Resultat

Redan våra första tester visade på 100% korrekta identifikationer. I vårt första test fick vår referens bestå av varsin ljudfil med en ungefärlig längd på 1 – 2 sekunder. Likaså var våra testfiler av ungefärlig samma längd. De ord vi spelade in både som referens och som testord, var helt godtyckligt valda, varför vårt resultat kan ses som en textoberoende klassificerare.

Det finns andra testrestultat på samma metod (enl. referens) som visar på cirka 96-98% träffsäkerhet. Vi vet dock inte exakt hur deras val av noggrannhet är. Denna noggrannhet borde bestå till stora delar i vilket antal kluster man väljer, där självfallet fler kluster innebär bättre upplösning. Empiriskt valde vi 8 kluster, vilket verkar fungera väldigt bra!

Vi har inte arbetat med någon optimering av de olika stegen i vår metod, utan följt rekommenderade inställningar. Man kan dock tänka sig att det går att snabba upp identifieringsprocessen genom att välja ett annat antal filter i filterbanken. Även antal kluster påverkar både snabbhet och noggrannhet.

Problem och möjliga förbättringar

Vi borde sätta in någon form av tröskelvärden för att även kunna avvisa personer som ej finns i referensdatabasen. Men vi väljer att se denna begränsning som ett designbeslut.
Ett annat problem som vi blev medvetna om är att en röst ej är statisk, utan ändrar sig över tiden. Till exempel låter man troligtvis annorlunda om man jämför sin röst med hur den lät för ett halvår innan.

Slutsatser

Både genom vårt egna genomförande av det här projektet och det vi läst på nätet så kan vi dra slutsatsen att om man går igenom de steg vi redovisat ovan så erhåller man en mycket bra metod för att klassificera och känna igen röster. Kommersiellt har en sådan metod definitivt framtiden för sig, speciellt vad gäller tendensen att människa-datorinteraktion allt mer går ifrån den traditionella mus-keyboard-modellen till en mer varierad interaktionsform. Även antalet inbyggda system ökar i alla branscher, där röstigenkänning med fördel kan användas. Vi har även sett att redan idag är detta möjligt till en rimlig datorkraft, varför det kanske inte är så avlägset som det till en början kan kännas. Som vi poängterade i början av rapporten är det endast fantasin som sätter gränser!

Appendix

Källkoder

(Vi har skrivit alla dessa funktioner förutom disteu(x, y), melfb(p, n, fs), som vi hittade på http://lcavwww.epfl.ch/~minhdo/asr_project/)

```
function test_project()

    filename = 'fabby1.wav'

    % Calculate the codebook vector.

cep = learn('supermannen1.wav');
    codebook = vqlbg(cep, 8);
    result(1) = identify(filename, codebook);

cep = learn('peter1.wav');
    codebook = vqlbg(cep, 8);
    result(2) = identify(filename, codebook);

cep = learn('mangan1.wav');
    codebook = vqlbg(cep, 8);
    result(3) = identify(filename, codebook);

cep = learn('fabby1.wav');
    codebook = vqlbg(cep, 8);
    result(4) = identify(filename, codebook);



% Test a file.

    if ( ( result(1) < result(2) ) & ( result(1) < result(3) ) & ( result(1) < result(4) ) )
    'Ted.'
    end

    if ( ( result(2) < result(1) ) & ( result(2) < result(3) ) & ( result(2) < result(4) ) )
    'Peter.'
    end

    if ( ( result(3) < result(2) ) & ( result(3) < result(1) ) & ( result(3) < result(4) ) )
    'Magnus.'
    end

    if ( ( result(4) < result(2) ) & ( result(4) < result(1) ) & ( result(4) < result(3) ) )
    'Fabyanna.'
    End
```

```
function dist = identify(file, codebook)

    [filedata, fs] = wavread(file);

    truncated = extract(filedata);

    cep = mfcc(truncated);

    d = disteu(cep', codebook);
    dist = sum(min(d,[],2)) / size(d,1);
```

```
function centroid = vqlbg(cepstrum, M)

% 000000000000000000000000000000000000000000000000
% ----------------------
% Feature matching
% ----------------------
% 000000000000000000000000000000000000000000000000

% STEP 1   --- initialization
centroid = medel(cepstrum);

centroid = centroid';  %'
cepstrum = cepstrum';  %'

%subplot(6,1,6); plot(centroid(1:resolution));
%title('The centroid.');

epsilon = 0.01;
%M = 8;                % The size of the codebook
m = 1;

while (m<M)

% STEP 2   --- division
temp_cent = centroid;
j = 1;
for i=1:m
   centroid(:,j)   = temp_cent(:, i) * (1 + epsilon);
   j = j + 1;
   centroid(:,j)   = temp_cent(:, i) * (1 - epsilon);
   j = j + 1;
end
m = 2*m;
test_var = 0;

D_prim = 99999999;      % Init D_prim to a sufficient large value.

while (test_var == 0)

% STEP 3   --- clustering
dist = disteu(cepstrum, centroid);
[magic, ind] = min(dist, [], 2);

% STEP 4   --- update
for i=1:m
%find(ind == i)
%i
%ind
centroid(:, i) = medel(cepstrum(:, find(ind == i))')';
end

% Calculate sum of distortions.
D = sum(sum(dist));

%( (D_prim - D) / D)
%epsilon
%m

if ( ( (D_prim - D) / D) < epsilon)
  test_var = 1;
else
  D_prim = D;
end

end

end
```

```
function d = disteu(x, y)
% DISTEU Pairwise Euclidean distances between columns of two matrices
%
% Input:
%       x, y:   Two matrices whose each column is an a vector data.
%
% Output:
%       d:      Element d(i,j) will be the Euclidean distance between two
%               column vectors X(:,i) and Y(:,j)
%
% Note:
%       The Euclidean distance D between two vectors X and Y is:
%       D = sum((x-y).^2).^0.5

[M, N] = size(x);
[M2, P] = size(y);

if (M ~= M2)
   error('Matrix dimensions do not match.')
end

d = zeros(N, P);

if (N < P)
   copies = zeros(1,P);
   for n = 1:N
      d(n,:) = sum((x(:, n+copies) - y) .^2, 1);
   end
else
   copies = zeros(1,N);
   for p = 1:P
      d(:,p) = sum((x - y(:, p+copies)) .^2, 1)';
   end
end

d = d.^0.5;
```

```
function cep = learn(file)

    [filedata, fs] = wavread(file);
    truncated = extract(filedata);

   cep = mfcc(truncated);
```

```
function sampledata = extract(xin, length)

% ----------------------------
%  Retrieving the sound data
%  and displaying it along with
%  its frequency representation.
% ----------------------------
%[xin, fz, length] = wavread(fname);


%subplot(6,1,1); plot( xin );
%title('Time domain analysis of the original wave.');

% ----------------------------
%  Extracting the most important
%  information in the sound
%  neglacting the noise by defining
%  a threshold.
% ----------------------------
mean = 0;

for i = 1:1000
   mean = mean + (abs(xin(i)) / 100);
end

threshold = mean * 2;

for first_index = 1:size(xin)
    if (abs(xin(first_index)) > threshold)
    break;
    end
end


for end_index = 1:size(xin)
    temp = size(xin) - end_index;
    if (abs(xin(temp(1))) > threshold)
    break;
    end
end

sampledata = xin(first_index:(size(xin) - end_index));
```

```
function m = medel(v);

[nr_of_rows, nr_of_columns] = size(v);


for i=1:nr_of_columns
m(i) = sum(v(1:nr_of_rows, i)) / nr_of_rows;
end
```

```
function m = melfb(p, n, fs)
% MELFB        Determine matrix for a mel-spaced filterbank
%
% Inputs:      p   number of filters in filterbank
%              n   length of fft
%              fs  sample rate in Hz
%
% Outputs:     x   a (sparse) matrix containing the filterbank amplitudes
%                  size(x) = [p, 1+floor(n/2)]
%
% Usage:       For example, to compute the mel-scale spectrum of a
%              colum-vector signal s, with length n and sample rate fs:
%
%              f = fft(s);
%              m = melfb(p, n, fs);
%              n2 = 1 + floor(n/2);
%              z = m * abs(f(1:n2)).^2;
%
%              z would contain p samples of the desired mel-scale spectrum
%
%              To plot filterbanks e.g.:
%
%              plot(linspace(0, (12500/2), 129), melfb(20, 256, 12500)'),
%              title('Mel-spaced filterbank'), xlabel('Frequency (Hz)');

f0 = 700 / fs;
fn2 = floor(n/2);

lr = log(1 + 0.5/f0) / (p+1);

% convert to fft bin numbers with 0 for DC term
bl = n * (f0 * (exp([0 1 p p+1] * lr) - 1));

b1 = floor(bl(1)) + 1;
b2 = ceil(bl(2));
b3 = floor(bl(3));
b4 = min(fn2, ceil(bl(4))) - 1;

pf = log(1 + (b1:b4)/n/f0) / lr;
fp = floor(pf);
pm = pf - fp;

r = [fp(b2:b4) 1+fp(1:b3)];
c = [b2:b4 1:b3] + 1;
v = 2 * [1-pm(b2:b4) pm(1:b3)];

m = sparse(r, c, v, p, 1+fn2);
```

```
function cepstrum = mfcc(x)

% -----------------------
% Frame blocking
% -----------------------


j=1;
i=1;
[s1, s2] = size(x);
%x(1: 256)

while ( (j+256) <= s1)
    for( k=1 : 256)
    x_new(i,k) = x(k+j-1);
    end
i = i + 1;
j = j + 156;
end

%subplot(2,1,1);
%plot(x_new(50,1:256))
%title('Plotting the 50th frame.')

% -----------------------
% Windowing
% -----------------------


j=1;
i=1;
[s1, s2] = size(x);
w = hamming(256);
%plot(w(1:256));
%title('Plotting the hamming-window.')


while ( (j+256) <= s1)
    for( k=1 : 256)
    x_new(i,k) = x_new(i,k) * w(k);
    end
i = i + 1;
j = j + 256;
end

%subplot(2,2,1);
%plot(x_new(50,1:256));
%title('Plotting the 50th frame (windowed).')


% -----------------------
% Fast Fourier Transform
% -----------------------

j=1;
i=1;

while ( (j+256) <= s1)
    x_new_freq(i,1:256) = fft(x_new(i,1:256));
    i = i + 1;
    j = j + 256;
end

%subplot(6,1,3);
%plot(abs(x_new_freq(25,1:128)))
%title('The 50th frame in the frequency domain.')


% -----------------------
% Mel frequency wrapping
% -----------------------

nr_of_filters = 20;
m = melfb(nr_of_filters, 256, 11000);
n2 = 1+floor(256/2);

i=1;
j=1;

while ( (j+256) <= s1)
    for (k=1:nr_of_filters)
    z_prim = (m * (abs(x_new_freq(i,1:n2)).^2)'); %'
z(i,k) = z_prim(k);
end
j = j + 256;
i = i + 1;
end

plot(z(1:20, 1:nr_of_filters));
title('Mel frequency wrapping.')

% -----------------------
% Cepstrum
% -----------------------

i=1;
j=1;

while ( (j+256) <= s1)
cepstrum_prim = dct(z(i,1:nr_of_filters));
for (k=1:nr_of_filters)
cepstrum(i,k) = cepstrum_prim(k);
end
    j = j + 256;
i = i + 1;
end

%cepstrum = cepstrum'; %'
resolution = i-1;

%subplot(6,1,5);
%plot(cepstrum(1:resolution, 1:nr_of_filters));
%title('The cepstrum.')
```

Referenser

http://www.ling.lu.se/persons/Mechtild/gbfiles/aastvt01/tikurs2000b.pdf
http://lcavwww.epfl.ch/~minhdo/asr_project/