Université de Toulouse III

Traitement Automatique de la Parole

Système de vérification de locuteurs

Auteurs :
M. Axel Bellec
M. Max Halford

Enseignant :
M. Jérôme Farinas
M. Julien Pinquier

Table des matières

1 Présentation de la base de données

Ecrire une fonction «lecture.m» permettant de lire (charger dans une variable) sous Matlab un fichier son et de connaître ses caractéristiques (fréquence d'échantillonnage, nombre de bits de quantification, durée).

```
function [signal, fs, nb_bits, duration]
↪  = lecture(fichier)
    [signal, fs] = audioread(fichier);
    nb_bits = 8;
    duration = nb_bits * fs;
end
```

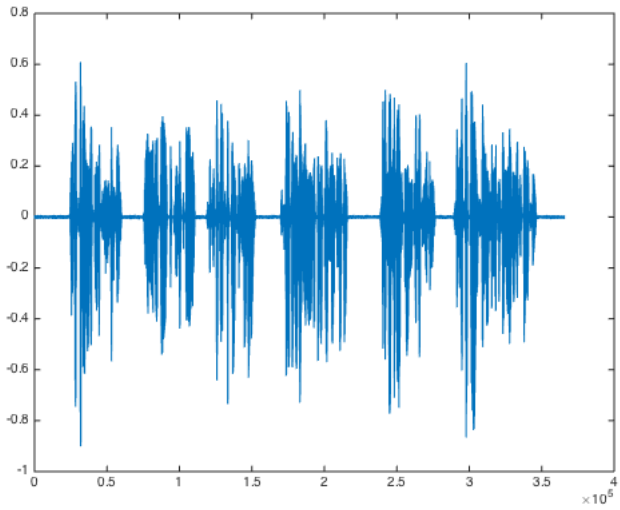

FIGURE 1 – Représentation graphique du signal

```
signal = lecture('APP/L1_fic1.wav');
plot(signal);
```

2 Décomposition parole/non-parole

Ecrire une première fonction «energie.m» permettant de calculer l'énergie à court terme d'un signal en fonction de la taille des fenêtres d'analyse.

Le signal étant représenté par un vecteur de longueur n, on va tout simplement boucler de 1 à n avec un pas de $\left\lfloor \frac{n}{p} \right\rfloor$ où p est le nombre d'intervalles voulus. Evidemment le dernier intervalle aura n mod p valeurs si n n'est pas un multiple de p.

```
function [energies] = energie(signal, taille_fenetre)
    energies = [];
    n = length(signal);
    pas = floor(taille_fenetre / 2);
    for i = 1:pas:n
        energie = sum(signal(i:(i+pas-1)) .^2) /
↪  taille_fenetre;
        %fprintf('from: %d | until: %d | energie:
↪  %d.\n', from, until, energie);
        energies = [energies, energie];
    end
end
```

Ecrire une autre fonction «etiquetage.m» (utilisant la fonction «energie») permettant d'étiqueter un fichier son en parole et non-parole sous Matlab.

Pour étiqueter un fichier son en zone de parole/non-parole, il suffit de parcourir le vecteur de l'énergie du signal. On regarde ensuite si pour chaque valeur d'énergie est supérieure à notre seuil. Si c'est le cas, on considère que c'est une zone de parole.

```
function [etiq_parole] = etiquetage(signal,
↪  taille_fenetre, threshold)
    etiq_parole = [];
    [energies] = energie(signal, taille_fenetre);
    for i=1:length(energies)
       if energies(i) >= threshold
           etiq_parole = [etiq_parole, 1];
       else
           etiq_parole = [etiq_parole, 0];
       end
    end
end
```

Ecrire une fonction «etiquetage_total.m» (utilisant la fonction «etiquetage») permettant d'étiqueter en parole/non-parole l'ensemble des fichiers sonores

du répertoire «WAV/APP».

On peut itérer à travers tout les fichiers du répertoire en remarquant qu'ils sont numérotés. On peut simplement faire boucler i de 1 au nombre de locuteurs et j de 1 au nombre de fichiers par locuteurs pour ensuite interpoler i et j afin d'obtenir le chemin de chaque fichier.

```
function etiquetage_total(taille_fenetre, nbe_loc,
↪  nbe_fic, threshold)
    basepath = 'APP';
    output_path = 'LABELS';
    % Loop through nbe_loc
    for i=1:nbe_loc
        % And then loop through nbe_fic
        for j=1:nbe_fic
            % Generate audio filename
            input_file = sprintf('%s/L%d_fic%d.wav',
↪  basepath, i, j);
            [signal, fs] = lecture(input_file);
            [etiq_parole] = etiquetage(signal,
↪  taille_fenetre, threshold);
            output_file = sprintf('%s/L%d_fic%d.lab',
↪  output_path, i, j);
            save(output_file,'etiq_parole','-ASCII');
        end
    end
end
```

3 Paramétrisation

Ecrire un programme permettant d'effectuer la paramétrisation (calcul des coefficients cepstraux).

Le cepstre est le résultat de la transformée de Fourier appliquée au logarithme naturel de la transformée de Fourier du signal dont la phase est ignorée. Ici on applique tout simplement le schéma classique d'extraction indiqué dans le TP.

```
function [mfcc] = parametrisation(signal,
↪  taille_fenetre, nbe_coef)
    mfcc = []; % Initialize matrix
    n = length(signal);
    pas = floor(taille_fenetre / 2);
    for i=1:pas:n
        cepstre_real =
↪  real(ifft(log(abs(fft(signal(i:(i+pas-1)) .*
↪  hamming(pas))))));
        calc_mfcc = cepstre_real(2:nbe_coef);
        mfcc = [mfcc; calc_mfcc']; % Transpose
↪  calc_mfcc
    end
end
```

Ecrire une fonction «parametrisation_total.m» (utilisant la fonction «parametrisation») permettant de calculer les MFCC pour l'ensemble des fichiers du répertoire «WAV/APP»

De la même manière que pour la fonction `etiquetage_total`, nous allons itérer sur les locuteurs i et leurs fichiers associés j pour calculer leur MFCC. Nous enregistrons ensuite la matrice obtenue dans un fichier.

```
function parametrisation_total(taille_fenetre, nbe_loc, nbe_fic, nbe_coef)
    basepath = 'APP';
    output_path = 'MFCC';
    % Loop through nbe_loc
    for i=1:nbe_loc
        % And then loop through nbe_fic
        for j=1:nbe_fic
            % Generate audio filename
            input_file = sprintf('%s/L%d_fic%d.wav', basepath, i, j);
            [signal, fs] = lecture(input_file);
            mfcc = parametrisation(signal, taille_fenetre, nbe_coef);
            output_file = sprintf('%s/L%d_fic%d.mfcc', output_path, i, j);
            save(output_file,'mfcc','-ASCII');
        end
    end
end
```

4 Apprentissage

Ecrire la fonction «apprentissage.m» permettant d'effectuer l'apprentissage des modèles. Cette fonction prendra en entrée les vecteurs de données à traiter (fichiers «monde» et «locuteur») ainsi que le nombre de lois gaussiennes n. Elle fournira en sortie les modèles du monde et du locuteur représentés par les poids w, les moyennes m et les variances v de chacune des lois gaussiennes

Ici on applique l'algorithme mis à disposition pour déterminer les paramètres des gaussiennes que l'on va utiliser pour modéliser le "monde" et les locuteur. Les paramètres sont d'abord approximés grâce à l'algorithme des K-means. Pour avoir des paramètres plus réalistes, l'algorithme EM est utilisé en calculant la probabilité d'appartenance de chaque locuteur à chaque distribution (*Expectation*) et en re-paramétrisant les distributions (*Maximization*).

```
1 function [m, v, w, m_l, v_l, w_l]
  ↪  = apprentissage(fic_monde, fic_locuteur, n)
2     monde = load(fic_monde);
3     [m, v, w] = gaussmix(monde, 1, 10.001, n);
4     locuteur = load(fic_locuteur);
5     [m_l, v_l, w_l] = gaussmix(locuteur, 1, 10.001,
  ↪  m, v, w);
6 end
```

5 Reconnaissance

Ecrire la fonction «tests_total.m» permettant d'effectuer la reconnaissance des fichiers sons inconnus situés dans le répertoire «WAV/RECO». Cette fonction prendra en entrée la taille de la fenêtre d'analyse, le nombre de coefficients cepstraux ainsi que les paramètres (poids, moyennes et matrices de covariance) de chacun des modèles, composés de mélanges de lois gaussiennes. Elle fournira en sortie un vecteur (ou matrice) du taux de reconnaissance par fichier.

On va comparer les probabilités d'appartenance pour un locuteur à la distribution de la gaussienne "monde" et à celle du locuteur. On utilise le MAP pour décider quelle distribution choisir (tout simplement on garde la probabilité retournée par chaque distribution la plus élevée).

```
function [taux_reco] = tests_total(taille_fenetre, nbe_coef, m, v, w, m_l, v_l, w_l)
    reco_path = 'RECO';
    taux_reco = [];
    for i=1:10
        for j=9:10
            input_file = sprintf('%s/L%d_fic%d.wav', reco_path, i, j);
            [signal, fs] = lecture(input_file);
            % Compute MFCC for test files
            mfcc = parametrisation(signal, taille_fenetre, nbe_coef);
            % Compute probabilities for each window
            prob_monde = gmmlpdf(mfcc, m, v, w);
            prob_locu = gmmlpdf(mfcc, m_l, v_l, w_l);
            % Create a new vector with 1 if locutor wins, else 0
            prob_tot = prob_monde < prob_locu;
            % Compute the sum for locutor
            locu_wins = sum(prob_tot);
            % Compute the final score
            score = locu_wins / length(prob_tot);
            taux_reco = [taux_reco, score];
        end
    end
end
```

6 Affichage des résultats

Nous avons développé une procédure supplémentaire pour afficher la classification que nous avons effectuée en fonction de la vrai classe.

```
function print_truth_predict(taux_reco)
    files = cell(20,1);
    locutor_truth = cell(20,1);
    basepath = 'RECO';
    index = 1;
    for i=1:10
        for j=9:10
            files{index} =
↪  sprintf('%s/L%d_fic%d.wav', basepath, i, j);
            locutor_truth{index} = sprintf('L%d', i);
            index = index+1;
        end
    end

    for i=1:length(taux_reco)
        if taux_reco(i) > 0.5
            disp(sprintf('[%s] Truth: %s - Predict:
↪  %s (score: %d)', files{i}, locutor_truth{i},
↪  'locutor', taux_reco(i)));
        else
            disp(sprintf('[%s] Truth: %s - Predict:
↪  %s (score: %d)', files{i}, locutor_truth{i},
↪  'monde', taux_reco(i)));
        end
    end
end
```

En exécutant cette procédure pour le locuteur n°1, nous obtenons :

```
Locutor tested: L1
[RECO/L1_fic9.wav] Truth: L1 - Predict: locutor (score: 5.538741e-01)
[RECO/L1_fic10.wav] Truth: L1 - Predict: locutor (score: 5.898953e-01)
[RECO/L2_fic9.wav] Truth: L2 - Predict: monde (score: 3.611663e-01)
[RECO/L2_fic10.wav] Truth: L2 - Predict: monde (score: 3.409091e-01)
[RECO/L3_fic9.wav] Truth: L3 - Predict: monde (score: 2.352941e-01)
[RECO/L3_fic10.wav] Truth: L3 - Predict: monde (score: 2.533003e-01)
[RECO/L4_fic9.wav] Truth: L4 - Predict: monde (score: 2.595238e-01)
[RECO/L4_fic10.wav] Truth: L4 - Predict: monde (score: 4.242301e-01)
[RECO/L5_fic9.wav] Truth: L5 - Predict: monde (score: 3.606838e-01)
[RECO/L5_fic10.wav] Truth: L5 - Predict: monde (score: 3.946416e-01)
```

```
[RECO/L6_fic9.wav] Truth: L6 - Predict: monde (score: 3.036131e-01)
[RECO/L6_fic10.wav] Truth: L6 - Predict: monde (score: 3.117563e-01)
[RECO/L7_fic9.wav] Truth: L7 - Predict: monde (score: 4.783198e-01)
[RECO/L7_fic10.wav] Truth: L7 - Predict: monde (score: 4.789562e-01)
[RECO/L8_fic9.wav] Truth: L8 - Predict: monde (score: 2.603858e-01)
[RECO/L8_fic10.wav] Truth: L8 - Predict: monde (score: 3.055556e-01)
[RECO/L9_fic9.wav] Truth: L9 - Predict: monde (score: 3.669951e-01)
[RECO/L9_fic10.wav] Truth: L9 - Predict: monde (score: 3.439364e-01)
[RECO/L10_fic9.wav] Truth: L10 - Predict: monde (score: 1.780899e-01)
[RECO/L10_fic10.wav] Truth: L10 - Predict: monde (score: 2.306080e-01)
```

7 Fonction main

Voici la fonction main qui résume toutes les opérations effectuées.

```
% Nombre de fichiers d'apprentissage et de tests.
nbe_loc = 10;
nbe_fic = 8;
nbe_coef = 32; % number of cepstral coefficient,
threshold = 0.0001;
taille_fenetre = 512; % frame width,

%% Preprocessing
% Labelize data
etiquetage_total(taille_fenetre, nbe_loc, nbe_fic,
↪  threshold);

% Compute MFCC
parametrisation_total(taille_fenetre, nbe_loc,
↪  nbe_fic, nbe_coef);

%% Learning

% Constants
nb_gauss = 4; % number of gaussian,
locutor = 1; % locuteur id

% Affectation step
affectation('LABELS', 'MFCC', nbe_loc, nbe_fic,
↪  locutor);

% Learning step
[m, v, w, m_l, v_l, w_l] =
↪  apprentissage('MFCC/monde.mfcc',
↪  sprintf('MFCC/L%d.mfcc', locutor), nb_gauss);

% Test total
taux_reco = tests_total(taille_fenetre, nbe_coef, m,
↪  v, w, m_l, v_l, w_l);
disp(sprintf('Locutor tested: L%d', locutor))
print_truth_predict(taux_reco);
```

Pour aller encore plus loin je pense que nous aurions pû mettre en oeuvre une stratégie Grid Search pour trouver les meilleurs paramètres pour améliorer la précision de notre classification. Les 2 paramètres les plus importants sont

le nombre de coefficients cepstraux à retenir lors du calcul de la MFCC, et le nombre de lois gaussiennes à utiliser lors de la phase d'apprentissage.