Reconocimiento de voz utilizando coeficientes mel-cepstrales

Pablo I. Chervonagura [1] *Jorge E. Gómez* [2] *Lucas A. Farré* [3] *Khalil Loukili* [4]

[1] ITBA - Instituto Tecnológico de Buenos Aires
pchervon@itba.edu.ar
[2] ITBA - Instituto Tecnológico de Buenos Aires
jogomez@itba.edu.ar
[3] ITBA - Instituto Tecnológico de Buenos Aires
lfarre@itba.edu.ar
[4] INSA Toulouse - Institut National des Sciences Appliquées de Toulouse
kloukili@etud.insa-toulouse.fr

Resumen

En este informe se presenta el método utilizado para el reconocimiento de una persona mediante el habla. Para ello, se utilizaron diferentes funciones desarrolladas en GNU Octave. Este sistema es capaz de leer un archivo de audio, analizarlo, y, utilizando un conjunto de voces pregrabadas, reconocer al individuo que generó el audio.

Palabras claves: Reconocimiento por voz, coeficientes cepstrales en las frecuencias de Mel (MFCC), métodos numéricos, cuantización vectorial, cepstrum

1 Introducción

El siguiente trabajo consiste en cumplir con el principal objetivo de desarrollar un sistema que permita identificar a un individuo analizando el espectro generado por una grabación.

Para lograr este objetivo se tomaron muestras de audio de doce personas diferentes, cinco grabaciones por individuo y estas se dividieron en dos categorías: datos (fragmento de prueba) y muestras a identificar (fragmento de entrenamiento).

Los datos consisten en un único archivo de audio por individuo, de los cuales se obtienen los coeficientes mel-cepstrales. Luego, utilizando un algoritmo de cuantización vectorial, se reduce la cantidad de muestras a dieciséis vectores código los cuales identifican a cada persona.

Las muestras a identificar consisten en cuatro archivos por individuo, de los cuales también se obtendrán los coeficientes mel-cepstrales, y mediante un algoritmo de búsqueda, se encuentra el conjunto de vectores código que minimizan la distorsión media de estos coeficientes. A partir de este resultado se obtiene que archivo de dato grabó el individuo del cual se tomó la muestra e identificando a la persona a la que pertenece dicha muestra.

2 Metodología

Para obtener las muestras de audio se contó con la colaboración de alumnos de Ingeniería en Informática del Instituto Tecnológico de Buenos Aires. Estas fueron grabadas utilizando el software de código abierto *Audacity*, a una frecuencia de muestreo de 8000 Hz y con un único canal de entrada (mono).

Inicialmente, se utiliza la función de GNU Octave *wavread* se obtiene la señal muestreada de los audios y la frecuencia de muestreo. Para obtener los coeficientes mel-cepstrales de los archivos de audio se utilizó como guía la publicación [1] y [2]. El procedimiento consiste de los siguientes pasos:

I. A la señal se le aplica un filtro de preénfasis para poder nivelar la señal del espectro. El preénfasis es el incremento del nivel de altas frecuencias de audio en proporción directa al aumento de amplitud del ruido en dichas frecuencias, antes de la modulación, con el fin de mantener una relación constante a través de toda la banda de transmisión [3]. Se utiliza la siguiente ecuación:

$$\square'_{\square} = \square_{\square} - \square\square_{\square-1}$$

Donde $\square$ es un coeficiente entre 0.9 y 1. Para el reconocimiento de voz generalmente se utiliza el valor 0.97.

II. Luego de pasar por el filtro, se divide la señal en pequeños fragmentos, *frames*. Para ello se toman *frames* de 20 ms de duración que contienen 160 muestras dentro, ya que la frecuencia de muestra es de 8000 Hz. Entre dos *frames* adyacentes se toman 10 ms de superposicionamiento para poder mantener un nivel estacionario entre ellas.

III. Debido a las grandes variaciones de cero a la señal y viceversa en los extremos de los *frames*, se les aplica la función de enventanado de Hamming a 160 puntos. Esto reduce el efecto de saltos en los bordes. La expresión de la ecuación es la siguiente:

$$\square\square\square(\square) = 0.54 - 0.46\square\square\square(2\square\frac{\square - 1}{\square - 1})$$

Donde *N* es la cantidad de puntos en cada *frame*, 160 en este caso, y *n* itera de 1 a *N*. Esta función la provee la biblioteca de GNU Octave.

IV. Para poder mejorar el reconocimiento por voz, se demostró empíricamente que es conveniente intentar aproximar el oído humano que recibe frecuencias no lineales a través del espectro del audio. Para ello, primero utilizando la Transformada Rápida de Fourier (Incluida en GNU Octave) se calculan las magnitudes de los espectros, y luego se les aplica filtros triangulares que son calculados por la función *filterbanks*.

V. Una vez aplicado el filtro, se obtienen los primeros 12 coeficientes cepstrales de la siguiente manera:

$$\square_{\square} = \sum_{\square=1}^{\square} (\square\square\square\ \square_{\square})[\square(\square - 0.5)\frac{\square}{\square}]$$

Donde $\square_{\square}$es la magnitud del *k*-ésimo elemento luego de haber aplicado el filtro. Para calcular el último coeficiente cepstral $\square_{13}$basta con calcular la energía del *frame* mediante:

$$\square = \square\square\square \sum_{\square=1}^{160} \square_{\square}^{2}$$

VI. Por último, calculamos los coeficientes *deltas* a partir de los 13 coeficientes utilizando:

$$\square\square_{\square} = \frac{1}{10}[\ 2(\square_{\square+2} - \square_{\square-2}) + (\square_{\square+1} - \square_{\square-1})\ ]$$

De esta manera, obtenemos un total de 26 coeficientes para cada *frame*.

VII. Ahora procedemos a reducir la cantidad de datos resultantes a solamente 16 vectores de 26 coeficientes cada uno, utilizando el algoritmo *vq* de cuantización vectorial brindado por la cátedra, donde la distorsión está dada por la suma de los cuadrados de los coeficientes calculados para cada *frame*. Cada persona estará identificada por estos vectores.

VIII. Finalmente, para determinar a qué individuo corresponde la grabación, buscamos cuál de los conjuntos de vectores minimiza la distorsión media. Para lograr esto, utilizamos el algoritmo *meandist*, también brindado por la cátedra.

2.1 Funciones

- ***main***

Función principal a ejecutar en Octave. Ejecuta el algoritmo y muestra la eficiencia del sistema. Además imprime por cada grabación, a quien corresponde y a quién se le adjudicó la identificación.

```
function main
	% Se leen las voces de entrenamiento y se guardan 16
vectores por voz
	f = readdir(strcat(pwd,'/Voces/Entrenamiento'));
	talkersNum = numel(f);
	for i = 3:talkersNum
		file = char(f(i));
		[s,fs] = wavread(strcat('./Voces/Entrenamiento/',
file));
		coef = mfcc(s,fs);
		vecCode(:,:,i-2) = vq(coef, 16);
		names{i-2} = substr(file, 1, -6);
	endfor

	talkersNum = talkersNum - 2;
	corrects = 0;

	% Se leen las voces de prueba y se las compara con las de
entrenamiento
	f = readdir(strcat(pwd,'/Voces/Prueba'));
	recordings = numel(f);
	for i = 3:recordings;
		file = char(f(i));
		[s,fs] = wavread(strcat('./Voces/Prueba/', file));
		coef = mfcc(s,fs);

		talker = substr(file, 1, -6);

		for j = 1:talkersNum
			mean = meandist(coef, vecCode(:,:,j));
			if(j == 1)
				min = [1, mean];
			endif
```

```
                        if(mean < min(2))
                                min = [j, mean];
                        endif
                endfor

                printf("La grabacion es de %s y se identifico a
%s\n", talker, names{min(1)})
                corrects = corrects + strcmp(names{min(1)}, talker);
        endfor

        recordings = recordings - 2;
        eff = (corrects / recordings) * 100
        printf("La eficiencia del sistema es del %f por ciento\n",
eff);
endfunction
```

- ***mfcc***

Esta función se encarga de obtener los coeficientes mel-cepstrales a partir de una señal y su correspondiente frecuencia de muestreo. Los pasos utilizados son aquellos descritos en la primera sección de *Metodología*.

```
function r = mfcc(sig, fs)
        % Se realiza el preénfasis
        coef = 0.97;
        sig = preemphasis(sig, coef);

        % Se fragmenta la señal
        frames = frame(sig, fs, 0.02, 0.01);

        % Se aplica la funcion ventana de Hamming
        samples = size(frames)(1);
        framesNum = size(frames)(2);
        h = hamming(samples);

        hframes = frames.*h;

        % Se calcula la transformada rápida de Fourier a cada
muestra
        % N = 2^nextpow2(samples);
        for i = 1:framesNum
                fftframes(:,i) = fft(hframes(:, i));
        endfor

        % Se multiplica por los filterbanks
```

```
        for i = 1:framesNum
                pframes(:,i) = (abs(fftframes(:,i)).^2);
        endfor

        fbanks = filterbanks(26, samples, fs, 300);

        half = samples/2 + 1;
        pr = fbanks * pframes(1:half, :);

        % Se obtienen los primeros 12 coeficientes cepstrum
        for i = 1:framesNum
                cn(:,i) = melcep(pr(:,i));
        endfor

        cn;

        % Se obtiene el 13º coeficiente
        for i = 1:framesNum
                cn(13,i) = logenergy(frames(:,i));
        endfor

        % Se calculan los coeficientes delta
        for i = 1:framesNum
                d = deltas(cn(:,i));
                for j = 1:13
                        cn(j+13,i) = d(j);
                endfor
        endfor

        r = cn;
endfunction
```

- ***preemphasis***

Realiza el pre énfasis de la señal enviada tomando el coeficiente obtenido como parámetro.

```
function res = preemphasis(signal, coef)
        res(1) = signal(1);
        for i = 2:length(signal)
                res(i) = signal(i) - coef * signal(i-1);
        endfor
endfunction
```

- ***frame***

Frame se encarga de tomar los fragmentos de una señal s, teniendo en cuenta la frecuencia de muestreo (fs), duración de un *frame* (fd) y solapamiento entre cada *frame* (fi). Termina devolviendo una matriz de (fd * fs) filas.

```
function r = frame(s, fs, fd, fi)
      N = fd * fs;
      % cantidad de muestras
      fstep = fi * fs;
      % step entre cada una
      M = floor((length(s) - N) / fstep + 1);

      indf = fstep * [ 0:(M-1) ];
      %índices por frames
    inds = [ 1:N ].';
      % indices por muestra
    ind = indf(ones(N,1),:) + inds(:,ones(1,M));

      r = s(ind);
endfunction
```

- ***filterbanks***

Función que calcula un banco de filtros triangulares recibiendo como parámetros la cantidad de filtros, el número de iteraciones de la Transformada Rápida de Fourier, la frecuencia de muestreo y la frecuencia mínima.

```
function fbank =
filterbanks(nfilt=26,nfft=512,samplerate=8000,lowfreq=300)
    %Primero es necesario pasar todas las frecuencias a unidades
Mels,
    %para ello se utilizan las funciones hztoml
    lowmel = hztoml(lowfreq);

    %La máxima frecuencia es la frecuencia de muestreo/2,
    %en este caso es 4000
    highmel = hztoml(samplerate/2);

    %Cantidad de puntos necesarios para la generación del banco
    npoints = nfilt + 2;

    %Como octave no dispone de la funcion para generar un vector
    %de una determinada cantidad de puntos, entre dos números.
    %Se debe calcular manualmente el paso
    step = (highmel-lowmel)/(npoints-1);
    melpoints = lowmel:step:highmel;
```

```
    %Pasa los elementos de cada vector a hz, luego
    %los multiplica por nfft y los divide por la frecuencia de
muestreo
    bin = floor((nfft+1)*arrayfun(@mltohz,melpoints)/samplerate);

    fbank = zeros(nfilt,nfft/2+1);

    %Realiza las iteraciones necesarias, aplicando
    %la función partida descripta en el paper.
    for j = 1 : nfilt
        fstart = bin(j);
        fend = bin(j+1);
        for k = fstart:fend
            fbank(j,k) = (k - bin(j))/(bin(j+1)-bin(j));
        end
        fstart = bin(j+1);
        fend = bin(j+2);
        for k = fstart:fend
            fbank(j,k) = (bin(j+2)-k)/(bin(j+2)-bin(j+1));
        end
    end
endfunction
```

- ***hztoml***

  Función que modifica la unidad de un valor de Hz a Mel.

```
function m = hztoml(a)
      m = 1127 * log(1+a/700.0);
endfunction
```

- ***mltohz***

  Función que modifica la unidad de un valor de Mel a Hz.

```
function h = mltohz(a)
      h = 700*(10^(a/2595.0)-1);
endfunction
```

- ***melcep***

Se encarga de calcular los coeficientes cepstrum para cada frame, devolviendo doce coeficientes dado un frame determinado.

```
function cn = melcep(elem)
      K = rows(elem);
```

```
	for n = 1:12
		aux = 0;
		for k = 1:K
			aux = aux + log(elem(k,1)) * cos(n * (k -
0.5) * pi / K);
		endfor
		cn(n) = aux;
	endfor
	cn = cn';
endfunction
```

- ***logenergy***

Se encarga de calcular el decimotercer coeficiente que también se agrega a los doce anteriores.

```
function E = logenergy(samples)
	s = 0;
	for n = 1:160
		s = s + samples(n) ** 2;
	endfor
	E = log10(s);
endfunction
```

- ***deltas***

Calcula los coeficientes delta a partir de trece coeficientes cepstrum. Estos deben ser agregados a los existentes por cada frame.

```
function d = deltas(c)
	d = c;
	d(1) = (2*(c(3)-c(1)) + (c(2)-c(1))) / 10;
	d(2) = (2*(c(4)-c(1)) + (c(3)-c(1))) / 10;
	for t = 3:11
		d(t) = (2*(c(t+2)-c(t-2)) + (c(t+1)-c(t-1))) / 10;
	endfor
	d(12) = (2*(c(13)-c(10)) + (c(13)-c(11))) / 10;
	d(13) = (2*(c(13)-c(11)) + (c(13)-c(12))) / 10;
endfunction
```

3 Resultados y Conclusiones

Los resultados obtenidos fueron muy satisfactorios, pues obtuvimos una efectividad del 89.58%, calculada en base a 48 muestras diferentes.

Luego de numerosas pruebas llegamos a la conclusión de que se logró elaborar un algoritmo lo suficientemente preciso para el reconocimiento de un individuo mediante su voz. Esto superó ampliamente nuestras expectativas iniciales.

Es destacable que la efectividad depende de varios factores que no fueron considerados primordiales como la fidelidad de la grabación de las voces así como ruido externo que pueda interferir con el individuo, ya que lo fundamental fue lograr escribir un buen algoritmo, pero sin dudas con grabaciones más precisas se podría alcanzar una mayor eficiencia.

También podrían realizarse mayores y mejores estudios para aislar que factores alteran el reconocimiento y en qué nivel. Alterando la cantidad de muestras, el número de personas utilizadas, el espectro de voz, el dispositivo usado para grabar, etc.

Queda para el futuro, mejorar el algoritmo para poder aplicarlo en futuros proyectos, quizás reescribiendo el mismo en otro lenguaje de programación más escalable como *Java* o *Python.*

4 Bibliografía

[1] Wei Han, Cheong-Fat Chan, Chiu-Sing Choy, and Kong-Pang Pun. An efficient mfcc extraction method in speech recognition. In *Proceedings IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006.*, pages 4 pp.–, May 2006.

[2] James Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

[3] http://es.wikipedia.org/wiki/Pre%C3%A9nfasis, Wikipedia.