Fites 6

Generació de claus

Màster Interuniversitari en Seguretat de les Tecnologies de la Informació i Comunicació

Cristian Requena Barreda

CANVI D'IMPLEMENTACIÓ

A la darrera entrega es va localitzar una problemàtica pel que fa al tractament de l'espectre invers de l'espectre de so, també anomenat *cepstrum*.
Segons el paper de referència principal de Monrose et al. [1], les posicions inferiors d'aquest són afectades pel moviment dels articuladors bucals del so i les superiors ho són pel so vocàlic. Degut a l'alta variabilitat dels valors d'aquest espectre sonor, s'han revisat altres papers per cercar un altre tipus d'implementació que faciliti l'obtenció de valors estables. D'aquests, s'ha seleccionat el paper *Multi-speaker voice cryptographic key generation* [2], que a més a més destaca els punts febles de l'anterior durant la seva introducció.

Tot i què l'algorisme comparteix la part de captura i d'obtenció de l'espectre sonor, divergeix en quant al tractament posterior que en realitza. En efecte, en comptes d'obtenir l'espectre invers (*cepstrum*), aplica un preprocessament anomenat Mel Frequency Cepstral Coefficients (MFCC) a les mostres de so, obtenint, per totes elles, un vector de 13 dimensions, que caracteritza les bandes d'una manera més adient a la veu humana. Les freqüències sonores de la veu que distingeixen el timbre de la mateixa són tractades amb molt de pes, mentre que les freqüències a partir de 3500Hz (on la parla humana no arriba) no són rellevants.

Posteriorment, aquest algorisme fa un processament amb un motor de reconeixement de parla (Automatic Speech Recognition – ASR) per obtenir els fonemes captats. Aquesta part ha estat omesa perquè va més enllà de l'abast del projecte, de manera que es realitzarà la generació de valors per a un fonema.

DESENVOLUPAMENT

L'algorisme desenvolupat per realitzar el càlcul de l'espectre de so (transformada discreta de Fourier) continua essent emprat, i a més a més ha estat necessari implementar un altre algorisme per calcular els coeficients de Mel (MFCC).

Les dades d'entrada d'aquest algorisme són l'espectre de so de la mostra i el nombre de filtres a aplicar, mentre que la sortida és un vector de 13 posicions. A aquest vector, a més, se li aplica l'algorisme de la transformada discreta del cosinus[1], i se li descarten el primer element.

A partir de l'obtenció d'aquest conjunt de vectors (recordem que s'obté un per cada mostra de so), es realitzarà l'operació de càlcul necessària per obtenir un valor suficientment estable per a que sigui constant per a diferents repeticions.

Per a minimitzar l'efecte de l'amplitud del so, es realitza un càlcul previ a l'MFCC per a normalitzar l'espectre de so, de manera que les bandes s'amplifiquen o no en funció de la màxima amplitud present.

1 http://es.wikipedia.org/wiki/Transformada_de_coseno_discreta

```
public class MFCC {
        private int numMelFilters = 26;
        private int numCepstra;
        private double lowerFilterFreq = 80.00;
        private double samplingRate;
        private double upperFilterFreq;
        private int samplePerFrame;
        DCT dct;
        public MFCC(int samplePerFrame, int samplingRate, int numCepstra) {
                this.samplePerFrame = samplePerFrame;
                this.samplingRate = samplingRate;
                this.numCepstra = numCepstra;
                upperFilterFreq = samplingRate / 2.0;
                dct = new DCT(this.numCepstra, numMelFilters);
        }
        public double[] doMFCC(double[] fft) {
                      double bin[] = fft;
                int cbin[] = fftBinIndices();
                double fbank[] = melFilter(bin, cbin);
                double f[] = nonLinearTransformation(fbank);
                double cepc[] = dct.performDCT(f);
                return cepc;
        }
        private int[] fftBinIndices() {
                int cbin[] = new int[numMelFilters + 2];
                cbin[0] = (int) Math.round(lowerFilterFreq / samplingRate * samplePerFrame);
                cbin[cbin.length - 1] = (samplePerFrame / 2);
                for (int i = 1; i <= numMelFilters; i++) {
                        double fc = centerFreq(i);
                        cbin[i] = (int) Math.round(fc / samplingRate * samplePerFrame);
                }
                return cbin;
        }
        private double[] melFilter(double bin[], int cbin[]) {
                double temp[] = new double[numMelFilters + 2];
                for (int k = 1; k <= numMelFilters; k++) {
                        double num1 = 0.0, num2 = 0.0;
                        for (int i = cbin[k - 1]; i <= cbin[k]; i++) {
                           num1 += ((i - cbin[k - 1] + 1) / (cbin[k] - cbin[k - 1] + 1)) * bin[i];
                        }

                        for (int i = cbin[k] + 1; i <= cbin[k + 1]; i++) {
                           num2 += (1 - ((i - cbin[k]) / (cbin[k + 1] - cbin[k] + 1))) * bin[i];
                        }
                        temp[k] = num1 + num2;
                }
                double fbank[] = new double[numMelFilters];
                for (int i = 0; i < numMelFilters; i++) {
                        fbank[i] = temp[i + 1];
                }
                return fbank;
        }
        private double[] nonLinearTransformation(double fbank[]) {
                double f[] = new double[fbank.length];
                final double FLOOR = -50;
                for (int i = 0; i < fbank.length; i++) {
                        f[i] = Math.log(fbank[i]);
                        if (f[i] < FLOOR) f[i] = FLOOR;
                }
                return f;
        }
        private double centerFreq(int i) {
                double melFLow, melFHigh;
                melFLow = freqToMel(lowerFilterFreq);
                melFHigh = freqToMel(upperFilterFreq);
                double temp = melFLow + ((melFHigh - melFLow) / (numMelFilters + 1)) * i;
                return inverseMel(temp);
        }
        private double inverseMel(double x) {
                double temp = Math.pow(10, x / 2595) - 1;
                return 700 * (temp);
        }
        protected double freqToMel(double freq) {
                return 2595 * log10(1 + freq / 700);
        }
        private double log10(double value) {
                return Math.log(value) / Math.log(10);
        }
```

Taula 1: Implementació MFCC.

```
public class DCT {

	int numCepstra;
	int M;
	public DCT(int numCepstra, int M) {
		this.numCepstra = numCepstra;
		this.M = M;
	}

	public double[] performDCT(double y[]) {
		double cepc[] = new double[numCepstra];
		for (int n = 1; n <= numCepstra; n++) {
			for (int i = 1; i <= M; i++) {
			    cepc[n - 1] += y[i - 1] * Math.cos(Math.PI * (n - 1) / M * (i - 0.5));
			}
		}
		return cepc;
	}
}
```

Taula 2: Implementació DCT.

```
	private static void debugValorsSeed(Integer[] seed) {
		if (seed[2]>6 && seed[2]<18 &&
			seed[3]>-5 && seed[3]<6 &&
			seed[4]>-10 && seed[4]<0 &&
			seed[5]>-10 && seed[5]<-3 &&
			seed[6]>-8 && seed[6]<-1 &&
			seed[7]>-5 && seed[7]<2 &&
			seed[8]>-2 && seed[8]<2 &&
			seed[9]>=0 && seed[9]<4 &&
			seed[10]>-3 && seed[10]<1 &&
			seed[11]>-3 && seed[11]<1) {
			System.out.println("S'ha reconegut la clau 'A'.");
		}
		if (seed[2]>5 && seed[2]<14 &&
			seed[3]>-1 && seed[3]<10 &&
			seed[4]>6 && seed[4]<15 &&
			seed[5]>-7 && seed[5]<18 &&
			seed[6]>2 && seed[6]<9 &&
			seed[7]>-6 && seed[7]<1 &&
			seed[8]>-13 && seed[8]<-3 &&
			seed[9]>-10 && seed[9]<-3 &&
			seed[10]>-8 && seed[10]<0 &&
			seed[11]>-6 && seed[11]<3) {
			System.out.println("S'ha reconegut la clau 'E'.");
		}
		if (seed[2]>5 && seed[2]<16 &&
			seed[3]>8 && seed[3]<22 &&
			seed[4]>13 && seed[4]<30 &&
			seed[5]>8 && seed[5]<30 &&
			seed[6]>3 && seed[6]<14 &&
			seed[7]>-10 && seed[7]<4 &&
			seed[8]>-10 && seed[8]<6 &&
			seed[9]>-10 && seed[9]<8 &&
			seed[10]>-1 && seed[10]<7 &&
			seed[11]>-2 && seed[11]<2) {
			System.out.println("S'ha reconegut la clau 'I'.");
		}
	}
```

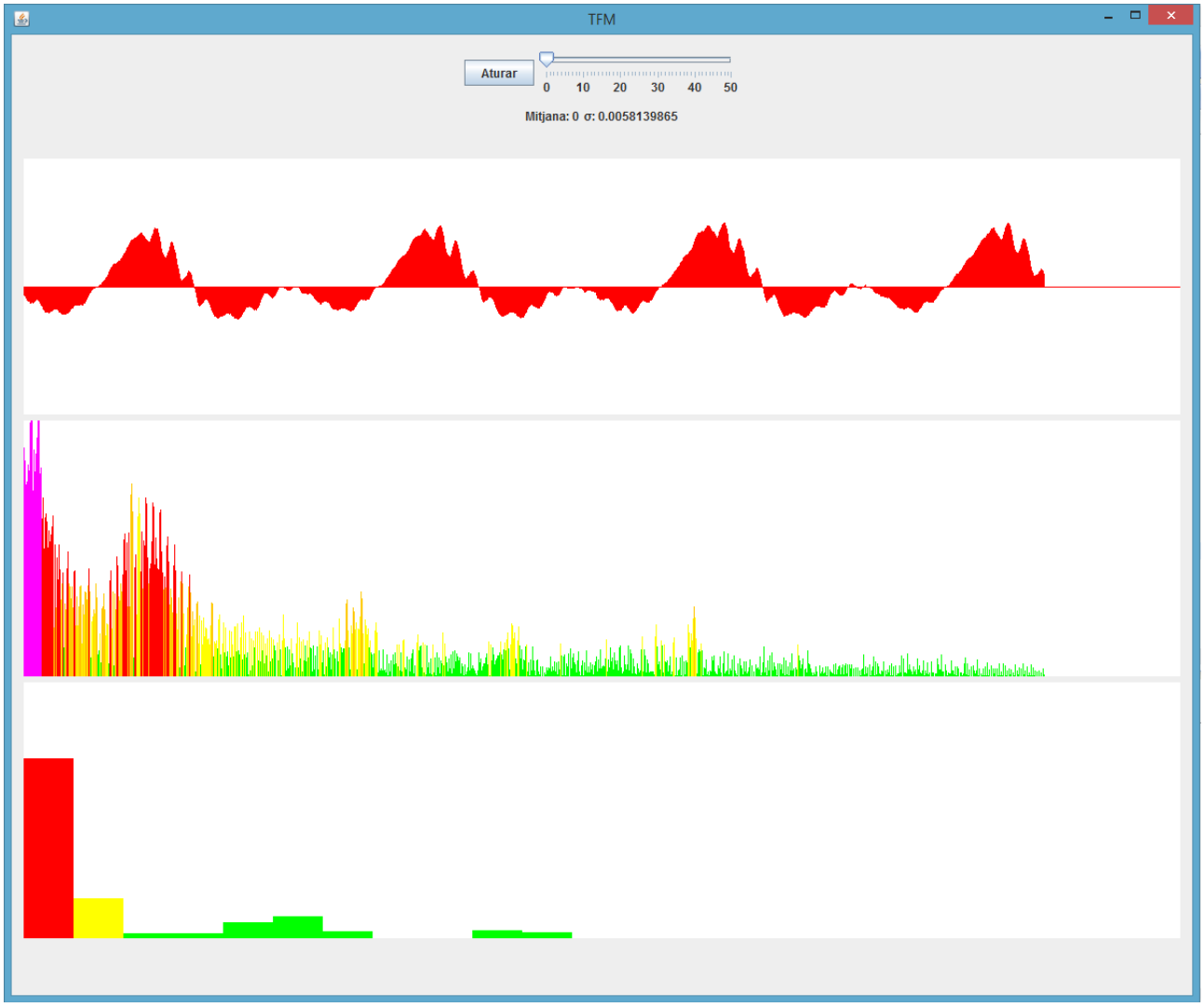
Taula 3: Algorisme per a fer proves de reconeixement de fonemes.

```
	private static void normalitzarEspectre() {
		// 1. Cercar el valor mínim/màxim de l'espectre.
		double min=999.0, max=-1.0;
		int minPos=-1, maxPos=-1;
		for (int i=0;i<espectre.length;i++) {
			if (espectre[i]>max) {
				max = espectre[i];
				maxPos = i;
			}
			if (espectre[i]<min) {
				min = espectre[i];
				minPos = i;
			}
		}

		// 2. Obtenir el factor de normalització
		double factor = 1048633.4400395132 / espectre[maxPos];

		// 3. Aplicar el factor de normalització a tot l'espectre.
		for (int i=0;i<espectre.length;i++) {
			espectre[i] = espectre[i] * factor;
		}
	}
```

Taula 4: Algorisme de normalització de l'espectre.

JOC DE PROVES

Visualment s'ha variat el tercer gràfic, que passa de ser el *cepstrum* a mostrar les 13 bandes Mel (MFCC).



S'han realitzat proves de reconeixement de veu amb diversos fonemes, obtenint diferents rangs de valors de les bandes Mel. A la següent taula es poden consultar els rangs, així com la relativa estabilitat, en comparació amb l'anterior implementació.

A										
15	6	13	2	-1	-8	-2	-1	0	0	0
16	4	10	0	-2	-6	-4	-3	0	1	0	0
16	5	9	1	-2	-4	-5	-1	-1	0	-2	0
12	10	17	5	-4	-9	-6	-1	0	1	-1	0
13	12	13	0	-7	-7	-7	-4	-1	3	0	-2
13	11	12	-3	-8	-6	-4	0	0	3	0	-2
13	10	14	0	-8	-11	-9	-1	0	2	-1	0
13	10	14	0	-6	-10	-8	-1	0	3	0	0
14	8	12	-1	-6	-7	-7	0	0	3	0	0
E										
12	8	12	4	11	9	7	-1	-6	-5	-2	1
14	8	8	3	9	11	3	-2	-9	-4	-4	0
14	8	6	0	8	14	5	-5	-9	-5	-5	-2
13	9	9	3	12	15	3	-3	-9	-4	-6	-2
12	11	12	13	16	15	6	-3	-10	-4	-3	-1
12	12	11	9	14	17	6	-6	-12	-6	-3	-1
12	11	10	7	15	19	6	-8	-14	-7	-3	-1
12	11	10	6	15	17	6	-7	-13	-6	-2	0
I										
11	9	11	15	22	17	6	2	3	6	2	0
11	10	11	13	21	19	7	0	-1	3	0	0
12	7	11	11	16	14	5	1	-1	3	0	0
12	9	11	12	18	17	6	0	-1	2	0	0
O										
14	7	15	13	9	2	-3	-4	-5	0	-1	0
13	10	18	17	8	1	-6	-5	-7	0	-2	1
12	10	15	13	6	0	-7	-8	-7	-1	-3	-2
13	8	15	12	6	0	-5	-2	-3	0	-2	-1
U										
14	6	13	12	8	5	0	2	-1	0	-2	-1
14	6	12	9	9	6	0	3	-1	1	-1	0
11	10	16	18	14	9	4	2	0	0	-3	-1
12	9	16	16	13	8	3	2	0	0	-3	0
12	8	14	16	12	7	4	4	0	2	0	0

Taula 5: Valors de les 12 bandes Mel per a diferents fonemes.

BIBLIOGRAFIA

[1]: Monrose, et al. (2001). Cryptographic Key Generation from Voice. *IEEE Symposium on Security and Privacy*.
[2]: L. Paola Garcia-Perera, J. Carlos Mex-Perera and Juan A. Nolazco-Flores (2005). Multi-speaker voice cryptographic key generation.