CAPITULO IV: Resultados

4.1 Pruebas

Se realizaron dos pruebas para determinar si el algoritmo funcionaba o no. La primera prueba se realizó sin un método de dependencia de características y el segundo con una dependencia condicional. Se explicará primero el algoritmo sin árboles de dependencia pues fue el primer algoritmo desarrollado para identificar a los hablantes. En ambos casos ocupamos los coeficientes Wavelet para estudiar las señales y determinar una expresión gráfica de las características extraídas de todas las señales procesadas. Esto con la finalidad de obtener una identificación de las señales.

El algoritmo llamado VoiceRecognition_2.py comienza con la lectura de los archivos de audio en un formato .wav de todas las señales a analizar (Figura22), pues era un formato con el que existe compatibilidad para poder trabajar con los métodos sugeridos. Además el formato .wav nos ofrece características importantes de calidad en una grabación para poder tener una señal lo suficientemente bien estudiada. Posteriormente se emplea el método pywt.dwt para extraer los coeficientes Wavelet CA y CD (Figura 23) propios de cada señal.

```
37 #_________________LECTURA AUDIO_COEFICIENTES WAVELET_________
38 #_____________Coef. de aproximaci?n y de detalle___________
39
40 fs, dataarray = wavfile.read('AnitaEmi.wav')                #5
41 pywt.dwt(dataarray, 'haar', 'sym')
42 (cA, cD) = pywt.dwt(dataarray, 'haar', mode ='sym')
43
44 fs, dataarray2 = wavfile.read('AnitaNat.wav')               #5
45 pywt.dwt(dataarray2, 'haar', 'sym')
46 (cA2, cD2) = pywt.dwt(dataarray2, 'haar', mode='sym')
```

Figura 22: Lectura del archivo .wav, conversión a arreglo numérico y extracción de coeficientes

Figura 23: Se muestra en la figura dos señales de voz diciendo lo mismo.

Posteriormente se realiza un modelo de mezcla de distribución normal mixture.NormalDistribution() (Figura 24) donde se realiza un modelo a partir de los coeficientes Wavelet para poder ser convertidos en datos leídos más adelante por el algoritmo Expectation Maximization (EM). Los nuevos datos se llaman nCA y nCD.

```
nCA = mixture.NormalDistribution(cA.mean(),cA.std())
nCD = mixture.NormalDistribution(cD.mean(),cD.std())
m = mixture.MixtureModel(2,[0.5,0.5], [nCA,nCD])
```

Figura 24: Modelo de mezcla para distribución normal

El siguiente paso que se realzó fue el de crear un arreglo que contuviera tanto los Coeficientes de aproximación y los de detalle. Para esto fue necesario convertirlos en una lista numérica y pegarlas con el comando list.extend() y posteriormente volverlos a unir como un arreglo numpy np.asarray(list). Inmediatamente (Figura 25) se muestra como ese nuevo arreglo

nombrado dataCACD es guardado en la variable dataM, pero ahora como un DataSet para poder ser introducido dentro del Expectation Maximization EM. Se le pusieron parámetros de tolerancia del 10% y un máximo de 40 pasos.

```
59 x = cA
60 x = np.array(x.tolist())
61 y = cD
62 y = np.array(y.tolist())
63 list = x.tolist()
64 list.extend(y.tolist())
65 dataCACD = np.asarray(list)
66
67 dataM = mixture.DataSet()
68 dataM.fromArray(dataCACD)
69 m.EM(dataM,40,0.1)
```

Figura 25: Algorítmo Expectation Maximization EM

Se debe resaltar que éste proceso se realiza para todas las señales que se involucren en el análisis del programa. Posteriormente se realiza el Expectation Maximization a cada señal y finalmente al imprimir m (Figura 26) nos genera un nueva curva de representación. Primero para los coeficientes de aproximación y luego los de detalle.

```
Parsing data set...done
Step 1: log likelihood: -437240.599968   (diff=-437239.599968)
Step 2: log likelihood: -405962.478585   (diff=31278.1213837)
Step 3: log likelihood: -402067.5894   (diff=3894.88918488)
Step 4: log likelihood: -401186.962624   (diff=880.626775298)
Step 5: log likelihood: -400853.224699   (diff=333.737925487)
Step 6: log likelihood: -400722.16756   (diff=131.057138682)
Step 7: log likelihood: -400669.469385   (diff=52.6981750657)
Step 8: log likelihood: -400647.864824   (diff=21.6045612023)
Step 9: log likelihood: -400638.88187   (diff=8.9829544322)
Step 10: log likelihood: -400635.110467   (diff=3.77140261477)
Step 11: log likelihood: -400633.51676   (diff=1.59370672697)
Step 12: log likelihood: -400632.840392   (diff=0.676368281944)
Step 13: log likelihood: -400632.552527   (diff=0.287865285063)
Step 14: log likelihood: -400632.429782   (diff=0.122745039233)
Step 14: log likelihood: -400632.377379   (diff=0.0524021516321)
Convergence reached with log_p -400632.377379 after 14 steps.

nCA2: Normal:  [-0.631838578892, 814.715076301]
nCD2: Normal:  [0.0176443334072, 89.4379245678]
[ 0.5  0.5]
```

Figura 26: EM alcanzando convergencia en 14 de 40 pasos e imprimiendo los nuevos coeficientes

Finalmente se grafican las curvas correspondientes para todas las señales ya con las componentes extraídas del algoritmo (mu, sigma, w) (Figura 27 y Figura 28). Inicialmente se encuentran muy pegadas pero al correr el algoritmo logra separarlas. Podemos observar que casi no existe separación en los coeficientes de aproximación, se observa también que muchas curvas se traslapan impidiendo así que haya una distinción de señales. Sin embargo, es notable que para los coeficientes de detalle es más fácil visualizar distinción entre las señales, a pesar de ello existen traslapes que no hacen tan efectiva la identificación.

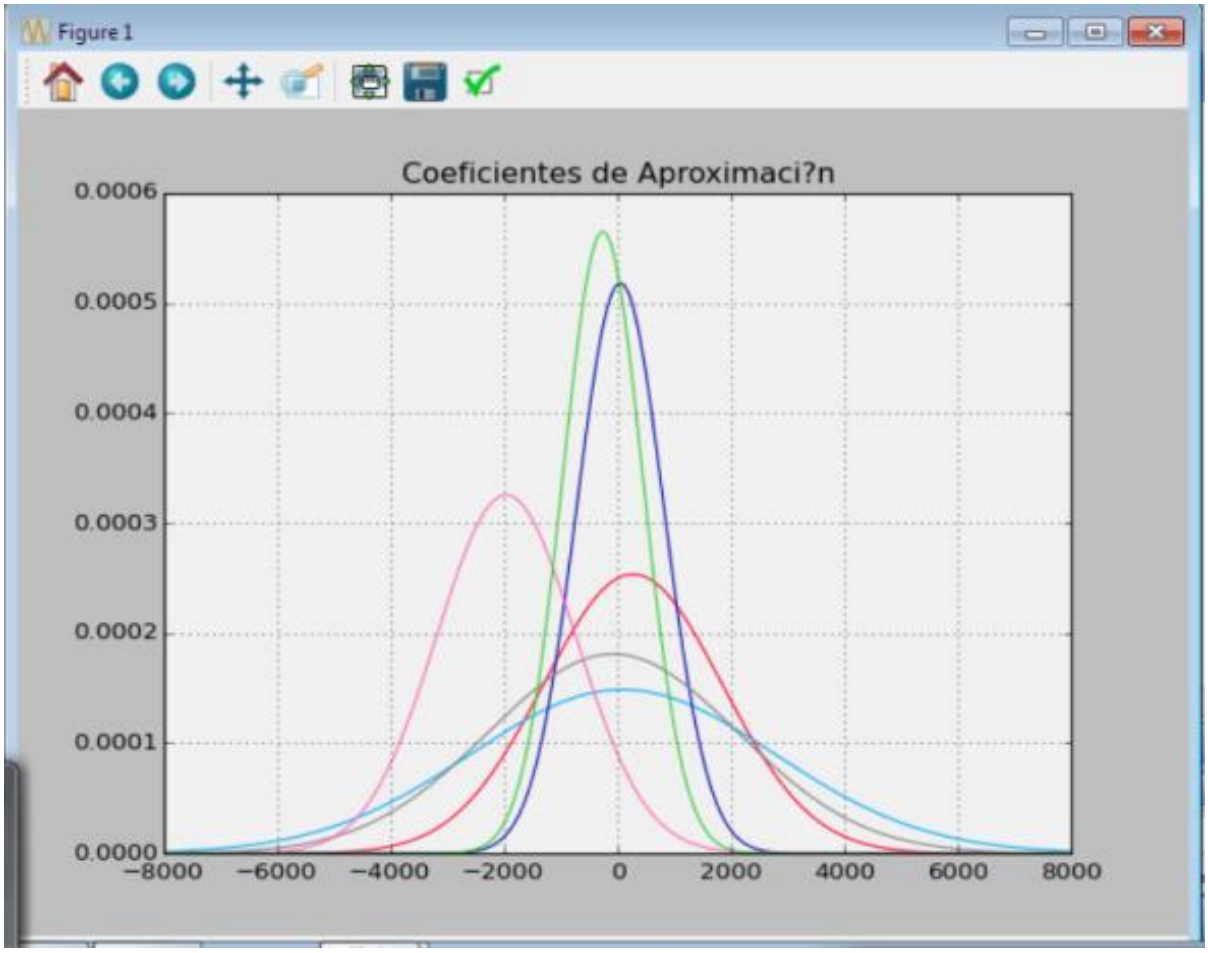

Figura 27: Gráficas de aproximación CA

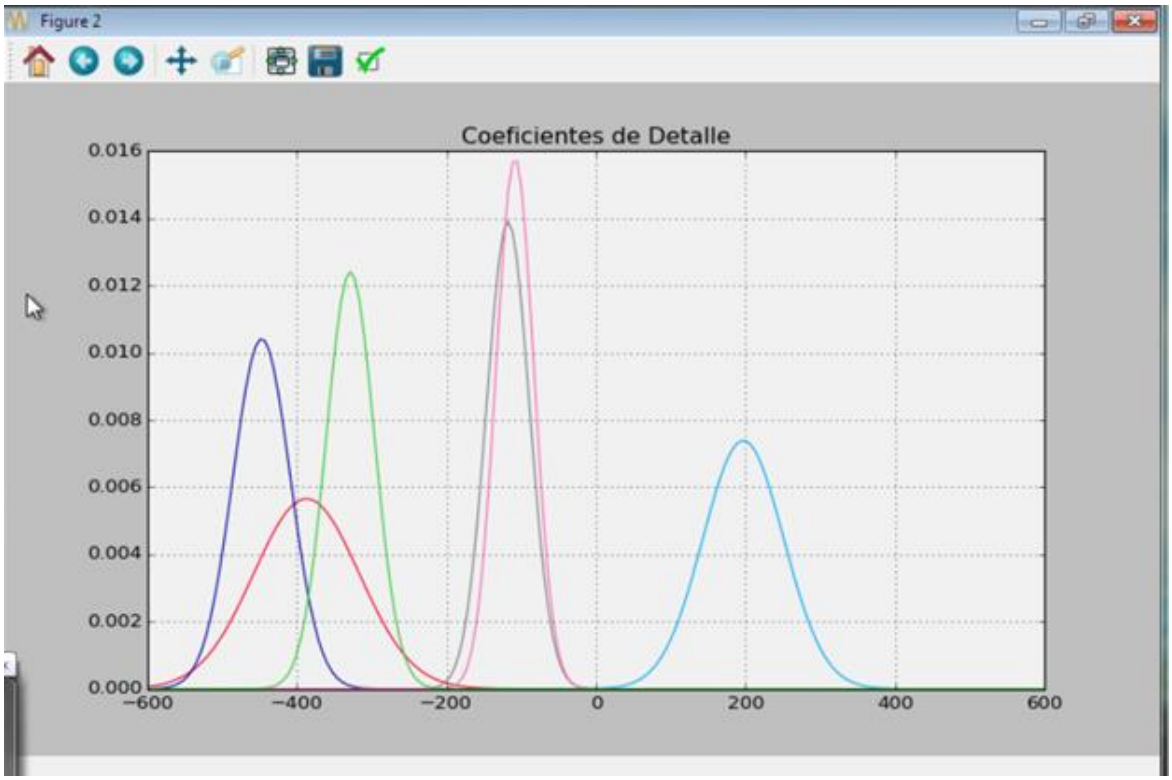

Figura 28: Graficación de curvas de coeficientes de detalle CD

Pudimos observar que la separación que se realizó para las curvas fue distinta, se observa que la separación no fue suficiente con las curvas de aproximación pues varias características de CA recaían en varias y observamos traslape. Por tal razón se realizó el mismo algoritmo pero en éste caso aplicando la teoría de árboles de dependencia, la cual nos dio mucho mejores resultados. Para éste otro algoritmo llamado DependenceTree_EXAMPLE.py se empieza de la misma manera que con el primer algoritmo primero convirtiendo el archivo de audio .wav en un arreglo numérico y posteriormente extrayendo los coeficientes Wavelet con la librería Pywt.

El arreglo de dependencia y de los árboles (Figura 29) se realizó configurando una dependencia de los coeficientes de detalle de los de aproximación. Todo el proceso se realizó de la misma manera utilizando arreglos de dos dimensiones donde se guardaba tanto la media como la desviación estándar.

```
#ARBOL 2
tree2 = {}
tree2[0] = -1
tree2[1] = 0
#n2 = mixture.ProductDistribution([mixture.ConditionalGaussDistribution(2,[cA2.mean(), cD2.mean()
n2 = mixture.ConditionalGaussDistribution(2,[cA2.mean(), cD2.mean()],[0, 5],[cA2.std(),cD2.std()]

#ARBOL 3
tree3 = {}
tree3[0] = -1    #-1 denota ra?z
tree3[1] = 0     # 0 denota siguiente rama, 1 la siguiente si existiera otro ?rbol.
n3 = mixture.ConditionalGaussDistribution(2,[cA3.mean(), cD3.mean()],[0, 5],[cA3.std(),cD3.std()]

#ARBOL 4
tree4 = {}
tree4[0] = -1    #-1 denota ra?z
tree4[1] = 0     # 0 denota siguiente rama, 1 la siguiente si existiera otro ?rbol.
n4 = mixture.ConditionalGaussDistribution(2,[cA4.mean(), cD4.mean()],[0, 5],[cA4.std(),cD4.std()]

#ARBOL 5
tree5 = {}
tree5[0] = -1    #-1 denota ra?z
tree5[1] = 0     # 0 denota siguiente rama, 1 la siguiente si existiera otro ?rbol.
n5 = mixture.ConditionalGaussDistribution(2,[cA5.mean(), cD5.mean()],[0, 5],[cA5.std(),cD5.std()]
```

Figura 29: Configuración de árboles de dependencia

Finalmente los resultados que arrojó el algoritmo fueron los siguientes (Figura 30). Se observa claramente que ahora ninguna curva se traslapa, podemos deducir que ahora la identificación de cada hablante no se compromete por curvas encimadas. En la tabla 1 podemos observar el órden y clasificación de las personas piloto que se estudiaron para el proyecto.

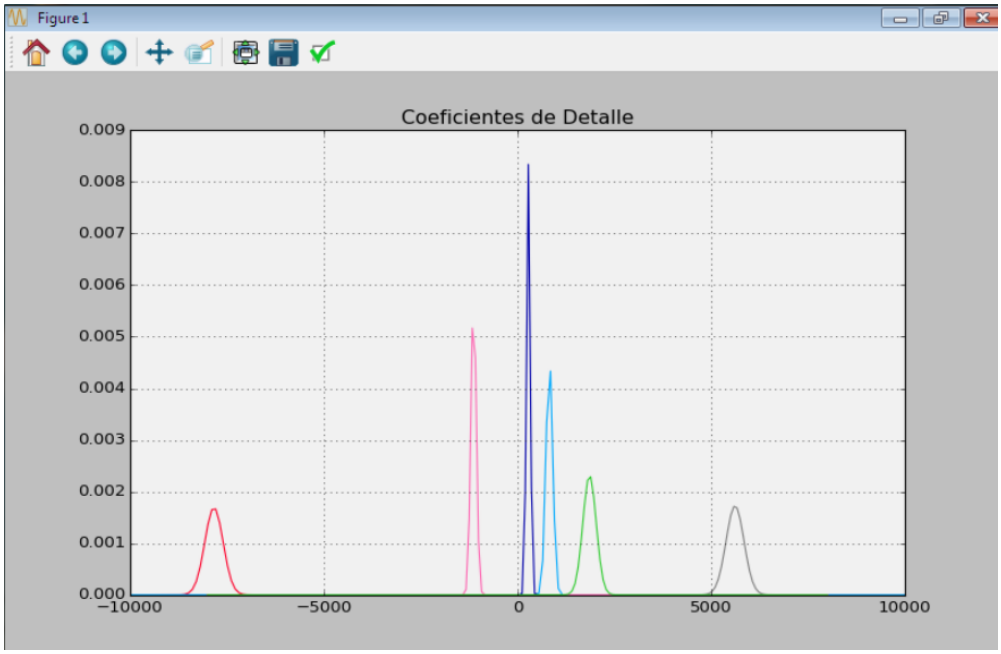

Figura 30: Curvas de detalle con árboles de dependencia.

No.	Color	Género	Frase
1.	Rojo	Hombre	“Anita lava la tina”
2.	Azul	Mujer	“Anita lava la tina”
3.	Azul Marino	Hombre	“Anita lava la tina”
4.	Gris	Mujer	“Anita lava la tina”
5.	Rosa	Hombre	“Anita lava la tina”
6.	Verde	Mujer	“Anita lava la tina”

Tabla 1: Pruebas realizadas a seis personas

Finalmente para poder comprobar la efectividad de este proyecto ocupamos la función clust(), para poder generar un cluster selectivo de características a partir de los muestras leídas de los modelos que generamos anteriormente. Para éste caso decidimos ocupar los datos de una sola persona, para que pudiéramos observar la diferencia en cuanto efectividad de reconocimiento tanto para un método como para el otro (con y sin árboles de dependencia). Por tal motivo se realizó inicialmente una prueba de clustering utilizando las muestras de una sola persona para probarlo con el algoritmo sin dependencia condicional. De este modo podríamos ver en cuantas otras muestras podrían recaer las características extraídas de una sola persona. El experimento se realizó dos veces y los resultados se muestran a continuación (Figura 31 y Figura 32).

Figura 31: Primer prueba de Cluster realizado al método sin árboles de dependencia

Para la primera prueba los datos que fueron utilizados por el clustering correspondían a la curva de color azul marino. Se puede observar claramente que a pesar de que existe un cluster que contiene el mayor número de muestras (perteneciente a la curva azul marino), es evidente que muchas de las muestras también recaen en otras personas, demostrando que la efectividad de reconocimiento sin utilizar árboles de dependencia carece de efectividad para poder discernir entre los hablantes al 100%. Caso similar ocurrió en la segunda prueba donde las muestras que se le dieron al clustering correspondían a la curva de color rosa.

Figura 32: Segunda Prueba realizada al método sin árboles de dependencia

Una vez que determinamos gráficamente a partir de las curvas características de los hablantes que el método de árboles de dependencia era más efectivo que el algoritmo sin éste método, decidimos realizar el clustering también, así, podríamos corroborar que el algoritmo utilizando árboles de dependencia las características extraídas recaerían completamente o mayormente en una sola persona. La prueba se realizó y se muestra a continuación (Figura 33) donde observamos que para 100 muestras recolectadas de una persona recayeron todas únicamente en ésta y no en otras personas.

Figura 33: Prueba de clustering para el algoritmo con árboles de dependencia

El mecanismo que utilicé para meter los datos de una sola persona al método de clustering, se realizó a partir de la creación de una mezcla a partir de los datos extraídos de una sola grabación para posteriormente extraer cierto número de muestras e incluirlas en el clustering. En la figura 34 podemos observar la arquitectura de ésta instrucción.

```
m2 = mixture.MixtureModel(2,[0.5,0.5],[nCA,nCD])
dataC = m2.sampleDataSet(100)
m2. modelInitialization(dataC)
#Clustering
clust = m.classify(dataC)
```

Figura 34: Estructura de inicialización de datos para clustering