Procesado de audio y voz

Práctica 4: Ampliación

Autores:
Èric Quintana
David Vizcarro

ÍNDICE

1. Introducción

En este apartado lo que intentaremos es mejorar los resultados obtenidos durante la parte básica de la práctica. Para ello intentaremos optimizar los parámetros de LP, LPCC y MFCC de distintas formas, extendiendo para ello el conocimiento que hemos adquirido en la primera parte de la tarea.

Para tomar un punto de referencia primero trataremos de realizar un sistema que minimice el error obtenido, sin tener en cuenta nada más. En caso de que los resultados no sean buenos, trataremos de analizar únicamente el coste.

2. Desarrollo:

2.1 Mejora LP

Para empezar, mejoraremos los coeficientes de LP, para ello analizaremos las distintas opciones que podemos aplicar:

```
patron@Ascalon:~/PAV/P4$ sptk help lpc

 lpc - LPC analysis using Levinson-Durbin method

  usage:
      lpc [ options ] [ infile ] > stdout
  options:
      -l l  : frame length                      [256]
      -m m  : order of LPC                      [25]
      -f f  : mimimum value of the determinant [1e-06]
            of the normal matrix
      -h    : print this message
  infile:
      windowed sequence (float)                [stdin]
  stdout:
      LP coefficients (float)
```

Como se ha podido comprobar en la primera parte del trabajo, lo más importante es el orden del LPC, ahora procederemos a investigar que sucede cuando alteramos:

- La longitud de la ventana
- El tipo de ventana
- La longitud del frame

Partimos de un error del 8,66% en la parte básica.

Ajustaremos la longitud de la ventana y el frame al mismo valor por el momento, partimos de un valor de 240:

Ventana ideal:

- Si bajamos todos los parámetros, con una ventana y frame de longitud 150 el error da 11,08%, por tanto ha subido.
- Si aumentamos todos los parámetros, con una ventana y frame de longitud 255, 270 el error da 7,77%, por tanto ha disminuido.
- Si aumentamos todos los parámetros, con una ventana y frame de longitud 350 el error da 8,28%, por tanto ha disminuido.
- Si aumentamos todos los parámetros, con una ventana y frame de longitud 500 el error da 9,17%.

Proseguiremos a partir de aquí con una longitud de 255, ahora probaremos cómo afecta cada tipo de ventana:

- 0 (Blackman) → 7,77%
- 1 (Hamming) → 8,92%
- 2 (Hanning) → 7,77%
- 3 (Bartlett) → 7,90%
- 4 (trapezoid) → 8,41%
- 5 (rectangular) → 7,13%

Proseguiremos a partir de aquí con una ventana rectangular, ahora probaremos cómo afecta tocar el periodo de la trama, partimos de un valor de 80:

- Si disminuimos el valor a 65, el error da 5,61%.
- Si disminuimos el valor a 50, el error da 3,95%.
- Si disminuimos el valor a 40, el error da 4,20%.

Proseguiremos a partir de aquí con un periodo de frame de 50, finalmente ajustaremos el orden del LPC, partimos de un valor de 15:

- Si aumentamos el valor a 20, el error da 4,71%.
- Si aumentamos el valor a 16, el error da 3,95%.
- Si disminuimos el valor a 14, el error da 4,59%.
- Si disminuimos el valor a 12, el error da 5,73%.

Parámetros finales:

```
EXEC="wav2lp 15 255 50 0 $db/$filename.wav $w/$FEAT/$filename.$FEAT"
```

2.1.1. Resultados LP (cambiando parámetros de ventana):

Comparación de parámetros:

	Ejecución original	Ejecución optimizada
Longitud de ventana	240	255
Longitud de trama	240	255
Periodo de trama	80	50
Tipo de ventana	Blackman	Rectangular
Coeficientes del LPC	15	15

Comparación de medidas de bondad:

	Ejecución original	Ejecución optimizada
Ratio de error	8,66%	3.95%
Coste de detección	61,9	64,4
Número de pérdidas	0.0010	0.6440
Falsas alarma	0.5200	0.0000
Umbral usado (TH)	2.60669	2.15495

Conclusión:

Tras haber hecho muchos tests, hemos conseguido un sistema que tiene muy poco error, pero tratando de minimizar el ratio de error hemos creado un sistema muy poco fiable puesto que el ratio de pérdidas es tremendamente elevado. Esto demuestra que quizá a veces es mejor no fijarse tanto en un elemento concreto sino en el cómputo global de los parámetros para ver si un sistema es realmente bueno o no. En este caso concreto hemos pasado de un sistema demasiado conservador a un sistema demasiado descuidado.

Por ello vamos a cambiar el enfoque y en vez de tratar de ajustar los parámetros de las ventanas, intentaremos bajar el coste y el error mediante el número de gaussianas.

Partimos de lo siguientes parámetros asociados al entrenamiento:

```
gmm_train    -v  1  -T  0.0001  -N  40  -m  100  -i  1  -d  $w/$FEAT  -e  $FEAT  -g
$w/gmm/$FEAT/$name.gmm $lists/class/$name.train || exit 1
```

```
gmm_train    -v  1  -T  0.0001  -N  50  -m  20  -i  1  -e  $FEAT  -d  $w/$FEAT  -g
$w/gmm/$FEAT/world.gmm $lists/verif/users_and_others.train  || exit 1
```

Para empezar partimos en el train con:

- Threshold: 0.0001
- Número de iteraciones: 40
- Número de Gaussianas: 100

Veremos secuencialmente cómo afecta cambiar cada parámetro a los resultados globales:

Para empezar ajustaremos el número de Gaussianas:

- 40: error 7.52%, coste 62.8
- **50: error 6,62%, coste 58.8**
- 55: error 7.39%, coste 59.1
- 60: error 7.77%, coste 57.1

Ahora ajustaremos el número de iteraciones manteniendo 50 Gaussianas:

- 15: error 6.11%, coste 66.0
- 25: error 6.62%, coste 60.4
- 30: error 6.75%, coste 60.4
- **50: error 6.11%, coste 58.8**
- 52: error 5.99%, coste 59.2

Ahora ajustaremos el threshold, manteniendo 50 Gaussianas y 50 iteraciones:

- 0.01: error 6.39%, coste 66.0
- 0.00001 error 6.11%, coste 58.8
- **0.0005: error 5.99%, coste 58.0**

Una vez tenemos el train podemos ajustar el trainworld para reducir el coste:

Para empezar ajustaremos el número de Gaussianas:

- 50: coste 67.9
- 30: coste 62.0
- **25: coste 57.2**
- 20: coste 58.0
- 15: coste 78.3

Ahora ajustaremos el número de iteraciones manteniendo 25 Gaussianas:

- 100: coste 60.0
- 75: coste 60.0
- 25: coste 60.7
- 15: coste 60

No ha habido mejoras por lo que seguiremos manteniendo las iteraciones en 50, ahora ajustaremos el umbral:

- 0.0005: coste 57.2
- 0.001: coste 57.2
- 0.01: coste 64.0

Probando umbrales tampoco hemos experimentado cambios por lo que los resultados para el LP quedan de la siguiente forma:

```
  gmm_train    -v  1  -T  0.0005  -N  50  -m  50  -i  1  -d  $w/$FEAT  -e  $FEAT  -g
$w/gmm/$FEAT/$name.gmm $lists/class/$name.train || exit 1
```

```
gmm_train    -v  1  -T  0.0001  -N  50  -m  25  -i  1  -e  $FEAT  -d  $w/$FEAT  -g
$w/gmm/$FEAT/world.gmm $lists/verif/users_and_others.train  || exit 1
```

2.1.2. Resultados LP (cambiando parámetros del train):

Comparación de parámetros:

Parámetros LP	Ejecución original	Ejecución optimizada
Longitud de ventana	240	240
Longitud de trama	240	240
Periodo de trama	80	80
Tipo de ventana	Blackman	Rectangular
Coeficientes del LPC	15	15
Parámetros train	**Ejecución original**	**Ejecución optimizada**
Umbral	0.0001	0.0005
Número iteraciones	50	50
Número de Gaussianas	100	20
Parámetros trainworld	**Ejecución original**	**Ejecución optimizada**
Umbral	0.0001	0.0001
Número iteraciones	50	50
Número de Gaussianas	20	25

Comparación de medidas de bondad:

	Ejecución original	Ejecución optimizada
Ratio de error	8.66%	6.62%
Coste de detección	61.9	57.2
Número de pérdidas	0.0010	0.5720
Falsas alarma	0.5200	0.0000
Umbral usado (TH)	2.60669	1.56241

Conclusión:

Tras intentar optimizar los valores del coste, vemos que no hemos podido reducir el error significativamente, pero observamos que nuevamente hemos hecho un sistema que tiene muchas pérdidas. Para tratar de minimizar el coste, realmente habría que tratar de hacer un mix entre la optimización de parámetros del LP y del train, pero esto resulta muy costoso puesto que los tiempos de ejecución son muy elevados (30-40 mins por ejecución).

2.2 Mejora LPCC

Nuevamente, mejoraremos los coeficientes de LPCC, para ello analizaremos las distintas opciones que podemos aplicar:

```
patron@Ascalon:~/PAV/P4$ sptk help lpc2c

 lpc2c - transform LPC to cepstrum

  usage:
      lpc2c [ options ] [ infile ] > stdout
  options:
      -m m  : order of LPC    [25]
      -M M  : order of cepstrum  [25]
      -h    : print this message
  infile:
      LP coefficients (float) [stdin]
  stdout:
      cepstrum (float)
```

Pero tal y como hemos visto con el LP, es mejor tratar de bajar el error y el coste de detección simultáneamente, por lo que iremos directamente a optimizar los parámetros del train. Partimos de lo siguientes parámetros asociados al entrenamiento:

```
gmm_train -v 1 -T 0.0001 -N 40 -m 100 -i 1 -d $w/$FEAT -e $FEAT -g
$w/gmm/$FEAT/$name.gmm $lists/class/$name.train || exit 1
```

```
gmm_train -v 1 -T 0.0001 -N 50 -m 20 -i 1 -e $FEAT -d $w/$FEAT -g
$w/gmm/$FEAT/world.gmm $lists/verif/users_and_others.train  || exit 1
```

Para empezar partimos en el train con:

- Threshold: 0.0001
- Número de iteraciones: 40
- Número de Gaussianas: 100

Veremos secuencialmente cómo afecta cambiar cada parámetro a los resultados globales:

Para empezar ajustaremos el número de Gaussianas:

- 40: error 0.76%, coste 20.3
- 50: error 1.27%, coste 18.4
- 60: error 0.89%, coste 18.0
- **65: error 1.02%, coste 11.6**
- 67: erro 0.89%, coste 13.2

Ahora ajustaremos el número de iteraciones manteniendo 65 Gaussianas:

- 15: error 0.89%, coste 15.2
- 20: error 0.89%, coste 14.4
- 30: error 1.02%, coste 16.4
- 50: error 1.02%, coste 11.6
- 70: error 1.02%, coste 14.4

Como podemos ver, 40 iteraciones da los mejores resultados, y aunque son iguales a 50 iteraciones, seleccionamos 40 por qué el tiempo de ejecución es menor.

Ahora ajustaremos el threshold, manteniendo 65 Gaussianas y 40 iteraciones:

- 0.01: error 1.02%, coste 18.8
- 0.00001: error 1.02%, coste 11.6
- 0.0005: error 1.02%, coste 16.4
- 0.008: error 0.89%, coste 18.4

Nuevamente en el threshold no tenemos una mejora clara por lo que continuaremos con el threshold a 0.0001.

Una vez tenemos el train podemos ajustar el trainworld para reducir el coste:

Partimos de:
- Threshold: 0.0001
- Número de iteraciones: 50
- Número de Gaussianas: 20

Para empezar ajustaremos el número de Gaussianas:

- 50: coste 7.2
- **45: coste 6**
- 40: coste 10
- 30: coste 13.6
- 25: coste 11.2

Ahora ajustaremos el número de iteraciones manteniendo 45 Gaussianas:

- 100: coste 6
- 75: coste 6
- 25: coste 5.2
- 20: coste 4.8
- **15: coste 4.8**

Ahora ajustaremos el umbral manteniendo 45 Gaussianas y 15 iteraciones:

- 0.0005: coste 4.8
- 0.001: coste 4.8
- 0.01: coste 4.8

No ha habido mejoras por lo que seguiremos manteniendo el umbral a 0.0001.
Los resultados finales se muestran a continuación.

```
gmm_train    -v  1  -T  0.0001  -N  40  -m  65  -i  1  -d  $w/$FEAT  -e  $FEAT  -g
$w/gmm/$FEAT/$name.gmm $lists/class/$name.train || exit 1
```

```
gmm_train    -v  1  -T  0.0001  -N  15  -m  45  -i  1  -e  $FEAT  -d  $w/$FEAT  -g
$w/gmm/$FEAT/world.gmm $lists/verif/users_and_others.train  || exit 1
```

2.2.1. Resultados LPCC:

Comparación de parámetros:

Parámetros LPCC	**Ejecución original**	**Ejecución optimizada**
Longitud de ventana	240	240
Longitud de trama	240	240
Periodo de trama	80	80
Tipo de ventana	Blackman	Blackman
Coeficientes del LPC	20	20
Orden cepstrum	20	20
Parámetros train	**Ejecución original**	**Ejecución optimizada**
Umbral	0.0001	0.0001
Número iteraciones	50	40
Número de Gaussianas	100	65
Parámetros trainworld	**Ejecución original**	**Ejecución optimizada**
Umbral	0.0001	0.0001
Número iteraciones	50	15
Número de Gaussianas	20	45

Comparación de medidas de bondad:

	Ejecución original	**Ejecución optimizada**
Ratio de error	1.40%	1.02%
Coste de detección	17.1	4.8
Número de pérdidas	0.001	0.048
Falsas alarma	0.072	0
Umbral usado (TH)	0.16192	-0.07989

Conclusión:

En este caso, al fijarnos más en todos los parámetros hemos podido comprobar como de mucho cambia cada valor en función de ciertos puntos. Primero, nos fijamos bien en los valores referentes al número de iteraciones y gaussianas. Sorprende el hecho de poder reducir el número a 65, reduciendo 35 iteraciones y consiguiendo así unos resultados sorprendentes. Nuevamente, solo hemos tocado train y trainworld sabiendo que se podría mejorar a costa de mucho tiempo de computación una reducción con los parámetros de lpcc. También hemos podido reducir en trainworld las iteraciones, aunque lo más importante ha sido ver que reduciendo éstas, tocaba aumentar el número de gaussianas para conseguir un valor óptimo de coste computacional, ya que nuestro sistema consigue un error de solamente un 1,02%, y para el coste la única manera de bajar de 5 ha sido mediante usar más gaussianas y menos iteraciones. Además, como punto importante, la probabilidad de falsa alarma la hemos bajado a 0,00.

2.3 Mejora MFCC

Para empezar partimos en el train con:

- Threshold: 0.0001
- Número de iteraciones: 40
- Número de Gaussianas: 100

Veremos secuencialmente cómo afecta cambiar cada parámetro a los resultados globales:

Para empezar ajustaremos el número de Gaussianas:

- 40: error 8.66%, coste 84.7
- 50: error 7.01%, coste 78.7
- 55: error 5.61%, coste 77.8
- 60: error 5.48%, coste 75.0
- **65: error 5.10%, coste 74.2**
- 70: error 5.73%, coste 79.0

Ahora ajustaremos el número de iteraciones manteniendo 50 Gaussianas:

- 15: error 4.84%, coste 81.0
- 25: error 5.35%, coste 81.0
- 30: error 5.48%, coste 80.6
- 50: error 5.73%, coste 74.2
- **60: error 5.61%, coste 73.8**
- 70: error 5.61%, coste 73.8

Ahora ajustaremos el threshold, manteniendo 65 Gaussianas y 50 iteraciones:

- 0.01: error 4.97%, coste 80.2
- 0.00001 error 5.86%, coste 73.8
- 0.0005: error 5.73%, coste 75.0

Una vez tenemos el train podemos ajustar el trainworld para reducir el coste:

Para empezar ajustaremos el número de Gaussianas:

- 70: coste 72.2
- 50: coste 70.2
- **40: coste 69.8**
- 30: coste 74.7
- 25: coste 74.6
- 20: coste 73.8
- 15: coste 89.0

Ahora ajustaremos el número de iteraciones manteniendo 40 Gaussianas:

- 100: coste 69.8
- 75: coste 69.8
- 25: coste 69.8
- **15: coste 69.4**

Ahora ajustaremos el umbral manteniendo 40 Gaussianas y 10 iteraciones:

- 0.0005: coste 70.2
- 0.000008: coste 69.4
- 0.001: coste 69.4
- 0.01: coste 71.4

No ha habido mejoras por lo que seguiremos manteniendo el umbral a 0.0001.
Los resultados finales se muestran a continuación.

```
gmm_train    -v  1  -T  0.0001  -N  60  -m  65  -i  1  -d  $w/$FEAT  -e  $FEAT  -g
$w/gmm/$FEAT/$name.gmm $lists/class/$name.train || exit 1
```

```
gmm_train    -v  1  -T  0.000008  -N  15  -m  40  -i  1  -e  $FEAT  -d  $w/$FEAT  -g
$w/gmm/$FEAT/world.gmm $lists/verif/users_and_others.train  || exit 1
```

2.3.1. Resultados MFCC:

Comparación de parámetros:

Parámetros MFCC	Ejecución original	Ejecución optimizada
Longitud de ventana	240	240
Longitud de trama	240	240
Periodo de trama	80	80
Tipo de ventana	Blackman	Rectangular
Orden MFCC	40	40
Parámetros train	**Ejecución original**	**Ejecución optimizada**
Umbral	0.0001	0.0001
Número iteraciones	50	60
Número de Gaussianas	100	65
Parámetros trainworld	**Ejecución original**	**Ejecución optimizada**
Umbral	0.0001	0.0001
Número iteraciones	50	15
Número de Gaussianas	20	40

Comparación de medidas de bondad:

	Ejecución original	Ejecución optimizada
Ratio de error	4.59%	5.61%
Coste de detección	75.0	69.4
Número de pérdidas	0.0020	0.496
Falsas alarma	0.5520	0.002
Umbral usado (TH)	1.98419	1.15853

Conclusión:

En esta versión optimizada hemos querido bajar el coste de detección anterior, cuyo valor era muy alto. La máxima mejora conseguida ha sido una reducción de 5,6 puntos dejándonos en 69,4 , lejos de ser un buen valor pero algo mejor. Las consecuencias de este cambio han sido un mayor ratio de error (aumentando un 1% el error), mayores pérdidas pero menores falsas alarmas. Comparando los valores, el simple hecho de haber conseguido bajar de 70 el coste de detección sin deteriorar excesivamente los otros valores nos hace dar por “buena” la optimización de mfcc.

3. Conclusión final:

Tras haber intentado optimizar los sistemas tenemos un claro “ganador”, el LPCC. Este sistema es el que tiene menos error y coste de entre todos lo que habíamos probado. Por otro lado aunque hemos podido mejorar el resto de sistemas, sentimos que los resultados obtenidos son bastante deficientes y que se podrían mejorar aún más con el suficiente tiempo y paciencia. Algunas de las mejoras adicionales que podríamos hacer son tratar de intentar cambiar los parámetros específicos de cada sistema como las ventanas y sus longitudes, viendo como el coste disminuye con cada cambio. También podríamos haber intentado mejorar el sistema en base al tiempo de ejecución, pero creemos que esto era un requisito secundario y por ende no lo hemos tenido en cuenta.