Universidad de Buenos Aires
Facultad De Ingeniería
Año 2018 - 1er Cuatrimestre

Modelo oculto de Markov, grámatica finita y HTK

Notas del Trabajo Práctico Final
Curso: Procesamiento del Habla (86.53)
Fecha: 12/11/2018
Docentes: Patricia Pelle, Claudio Estienne
Alumno:

Pérez Boco, Federico - #96777
<fedeboco@gmail.com>

1 Objetivos

A lo largo del presente proyecto se busca implementar un sistema de habla continua y de gramática finita mediante el uso de `HMM`, basado en el software `HTK`. Se buscará responder las siguientes preguntas:

1. **El problema de evaluación.** Dado un modelo Φ y una secuencia de observaciones $X = (X_1, ..., X_T)$ ¿cuál es la probabilidad $P(X|\Phi)$ de que el modelo genere dichas observaciones?
2. **El problema de decodificación.** Dado un modelo Φ y una secuencia de observaciones X ¿Cuál es la secuencia de estados más probable?
3. **El problema de aprendizaje.** Dado un modelo Φ y una secuencia de observaciones X ¿Cómo podemos ajustar el modelo Φ para maximizar la probabilidad conjunta $\prod_X P(X|\Phi)$?

2 Uso de HTK

2.1 Formato DARPA

El formato DARPA es el formato en el que se almacenan los datos de los n-gramas. Por ejemplo:

\1-grams
-0.89 </s> -1.089
-1.82 <s> -0.946
-3.59 a -0.476
-4.85 abajo -0.231
\2-grams
-2.29 <s> a
-3.12 <s> abrigamos
-4.55 <s> asustamos

Donde la primer columna hace referencia al logaritmo de la probabilidad de que ocurra dicho n-grama, la segunda columna es el n-grama en cuestión y en el caso de los 1-gramas la tercer columna es la función δ propia del método de *Back-off* de Kneser-Ney.

2.2 Opciones de funciones de HTK

`-A`	Print command line arguments
`-B`	Store output HMM macro files in binary
`-C cf`	Configuration file is cf
`-D`	Display configuration variables
`-F fmt`	Set source data file format to fmt
`-G fmt`	Set source label file format to fmt
`-H mmf`	Load HMM macro file mmf
`-I mlf`	Load master label file mlf
`-J tmf`	Load transform model file tmf
`-K tmf`	Save transform model file tmf
`-L dir`	Look for label files in directory dir
`-M dir`	Store output HMM macro files in directory dir
`-O fmt`	Set output data file format to fmt
`-P fmt`	Set output label file format to fmt
`-Q`	Print command summary info
`-S scp`	Use command line script file scp
`-T N`	Set trace level to N
`-V`	Print version information
`-X ext`	Set label file extension to ext

2.3 Base de datos

Para comenzar con el uso de HTK es necesario disponer de una base de datos. En nuestro caso utilizamos latino40.

>3 Mel Frequency Cepstral Coefficients (MFCC)

El procesamiento comienza con la codificación de los coeficientes cepstrum a partir de las señales de audio, generalmente en formato WAV. HTK posee una herramienta llamada HCopy capaz de realizar diversas tareas (copiar archivos por ejemplo). Entre ellas se encuentra la función de codificación de audio (.wav) a coeficientes cepstrum en escala mel (.mfc).

3.1 Analogía con TP5: Procesamiento Cepstrum

En el TP5 se discutió el procesamiento de un audio para obtener los **coeficientes mel cepstrum**. La escala mel es una escala que relaciona la percepción del oído humano de las frecuencias percibidas con las frecuencias que realmente se encuentran presentes y permite distinguir mejor bajas frecuencias. Se puede transformar de escala lineal a escala mel aplicando la siguiente transformación:

$$B(f) = 1125 \times \ln\left(1 + \frac{f}{700}\right) \tag{1}$$

Para obtener los coeficientes se aplica una serie de transformaciones:

1. DFT a fragmentos de señal ventaneados $x[n]$. Obtenemos DFT $X[k]$.
2. **Mapeamos la energía de la DFT a la escala mel.** Para ello, se procede multiplicando la energía de la DFT por la función triangular $H_m[k]$, donde m es el número de filtro.

   $$T[m] = |X[k]|^2 H_m[k], \quad 0 \leq m \leq M \tag{2}$$

   Siendo M la cantidad de filtros en el banco de filtros Mel.
3. Se suman las DFT's filtradas y se aplica el logaritmo.

   $$S(m) = \log\left(\sum_{k=0}^{N-1} T[m]\right) \tag{3}$$

   Siendo N la cantidad de puntos de DFT.
4. Finalmente se aplica la transformada coseno y se obtienen los **coeficientes MFCC**:

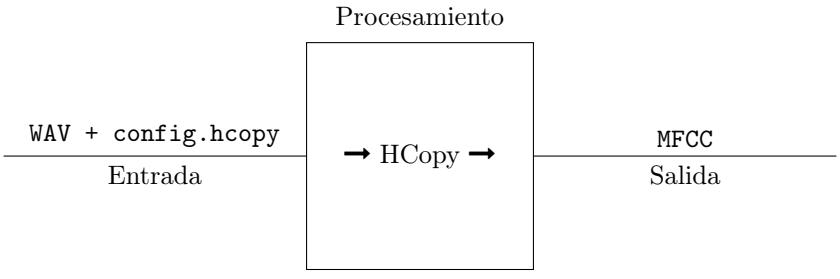
   $$c(n) = \sum_{m=0}^{M-1} s(m)\cos(\pi n(m + 0{,}5)/M), \quad 0 \leq n \leq M \tag{4}$$

3.2 HTK: Uso de HCopy

En esta sección se buscará generar los archivos MFCC (extensión `.mfc`) con los siguientes pasos:

- ✔ Generar carpetas para MFCC.
- ✔ Generar MFCC.

HCopy recibe como entrada los archivos de audio y un archivo de configuración.



Para generar las carpetas de salida usamos el script `go.mfclist`:

```
../scripts/go.mfclist genmfc.train genmfc.test
```

Cuyo contenido es:

```
#!/bin/sh
#Crea las listas archivos a procesar por HCopy
#y los directorios en mfc, Ejecutarlo desde datos
#

if [ $# -ne 2 ]
then
    echo Usar: $0 listatrain listatest
    exit 1
fi

ls wav/train |\
while read file
do
    mkdir mfc/train/$file
done
ls wav/test |\
while read file
do
    mkdir mfc/test/$file
done


ls wav/train/*/* >p
cat p |sed 's/wav/mfc/g' >q
paste p q > $1
rm p q

ls wav/test/*/* >p
cat p |sed 's/wav/mfc/g' >q
paste p q > $2
rm p q
```

Luego, HCopy se puede ejecutar como:

```
HCopy -A -V -T 1 -C ../config/config.hcopy -S genmfc.train
HCopy -A -V -T 1 -C ../config/config.hcopy -S genmfc.test
```

-S ("script") indica el origen y destino de los datos. `genmfc.train` Tiene la forma:

```
wav/train/fulano/fulano_001.wav mfc/train/fulano/fulano_001.mfc
wav/train/fulano/fulano_002.wav mfc/train/fulano/fulano_002.mfc
wav/train/mengano/mengano_001.wav mfc/train/mengano/mengano_001.mfc
wav/train/mengano/mengano_002.wav mfc/train/mengano/mengano_002.mfc
```

En el archivo config.hcopy podemos encontrar diversos parámetros como la cantidad de filtros m `NUMCHANS` a utilizar mencionados anteriormente.

SOURCEKIND	Formato de entrada
SOURCEFORMAT	Formato del header de las propiedades de archivo
SOURCERATE	Frecuencia a la que fue muestreado
TARGETKIND	Formato de salida
TARGETRATE	Frecuencia de muestreo de salida
WINDOWSIZE	Tamaño de ventana
NUMCHANS	Canales de banco de filtros ("m")
CEPLIFTER	Permite aplicar liftering
NUMCEPS	Cantidad de coeficientes de salida
USEHAMMING	Para usar ventana de Hamming
PREEMPHFILTER	Pasa altos al comienzo

3.3 Referencias HCopy

- Pág. 38, Sección 3.1.5 HTK Book.

>4 Diccionario

Para esta sección se requieren las transcripciones de los audios de la base de datos. Para latino40 son `promptsl40.train` y `promptsl40.test`.

Tienen la forma:

```
fulano_001      Los dos saben lo que es no poder agarrar una raqueta
fulano_002      y hacer lo que mejor saben hacer jugar al tenis
fulano_003      porque las lesiones obligan a parar y los dos
fulano_004      saben cuánto cuesta volver a empezar y resucitar de entre
fulano_005      las cenizas para regresar a esa elite a la que pertenecen
```

4.1 HTK: HDMan, Diccionario, lista de palabras y lista de fonemas

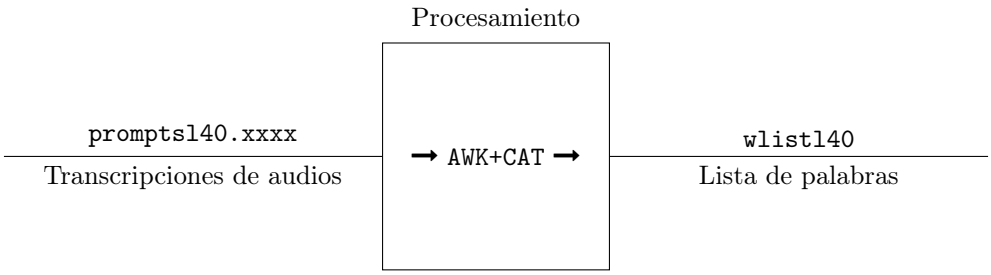
En el siguiente apartado procedemos a:

- ✔ Configurar Bash en español.
- ✔ Generar lista de palabras.
- ✔ Generar diccionario.
- ✔ Generar lista de fonemas.

1.1. Configurar Bash en español

```
export LC_CTYPE=ISO_8859_1
```

1.2. Lista de palabras



Para generar la lista de palabras se recurre a las herramientas `CAT` y `AWK`. Para entrenar el algoritmo suele utilizarse la totalidad de palabras tanto de train como de test, por ello se genera una lista con ambas. Una posible implementación es:

```
~/proyecto/etc$ cat promptsl40.train promptsl40.test |awk '{for(i=2;i<=NF;i++){print
    ↪ $i}}'|sort|uniq > wlistl40
```

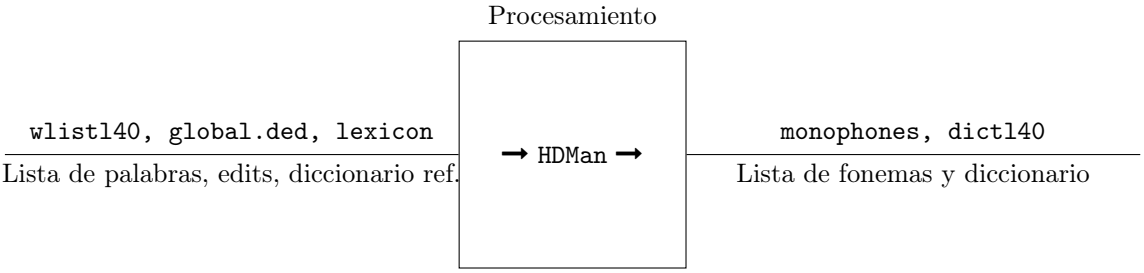
1.3. Diccionario

En primer lugar es necesario generar un archivo llamado global.ded cuyo contenido es:

```
AS sp
```

El mismo le indica a HDMan que debe agregar una pausa corta `sp` al final de cada pronunciación en el diccionario.

HDMan genera una lista de pronunciaciones (diccionario `dictl40`) y una lista de fonemas (`monophones`) a partir de una lista de palabras (`wlistl40`), el archivo (`global.ded`) y un diccionario de referencia (`lexicon`. En HTK Book son *Beep* y *Names*).



Obtenemos el diccionario y los fonemas:

```
HDMAn -w wlist40 -g global.ded -n monophones -l ../log/dlog dictl40 lexicon
```

El diccionario tiene la forma:

```
afecta          aa f ey k t aa sp
afectadas       aa f ey k t aa dd aa s sp
afectado        aa f ey k t aa dd ow sp
afectados       aa f ey k t aa dd ow s sp
```

1.4. Lista de fonemas

El archivo `monophones` generado no incluye los silencios. Para trabajar con cadenas de Markov es necesario incluirlos.

- Crear archivo `monophones1` editando `monophones` con un editor de texto y agregándole `sil` al final.
- Crear archivo `monophones0` editando `monophones1` con un editor de texto y borrándole `sp`.

4.2 Referencias HDMan

- Creación de diccionario: Pág. 25, sección 3.1 HTK Book.
- Expansión de global.ded: Pág. 173, sección 12.8 HTK Book.

>5 Master Label File (MLF)

En esta sección se explica como:

- ✔ Generar 1 Master Label File de palabras.
- ✔ Generar 1 Master Label File de fonemas.

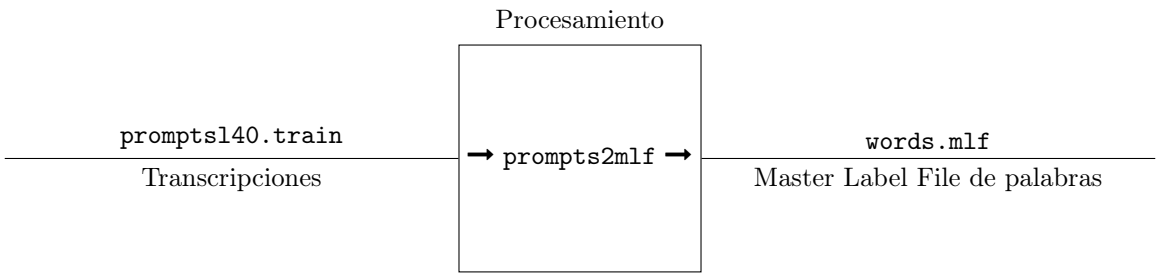
¿Qué función cumple un MLF? Un *label file* guarda la transcripción de palabras de un fragmento de audio con el mismo nombre de archivo pero con extensión `.lab`. El *master label file* es la concatenación de todos los *label files* y puede generarse directamente sin pasar por los archivos individuales, cómo se aplica en este trabajo.

Por ejemplo, un MLF de palabras tiene la forma:

```
#!MLF!#
"fulano_001.lab"
no
habiendo
objeciones
así
quedó
acordado
.
"fulano_002.lab"
a
fin
de
año
hará
el
balance
de
la
situación
.
"fulano_003.lab"
...
```

5.1 HTK: Master Label File de palabras

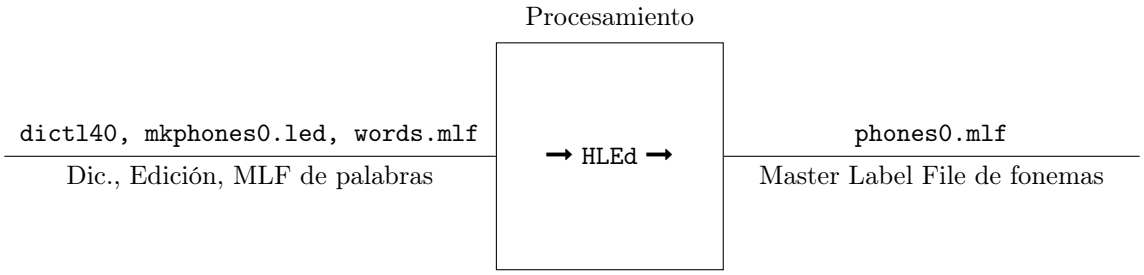
Para generar el MLF utilizamos el script `prompts2mlf` que genera el MLF a partir de una lista de transcripciones (que a su vez especifican el nombre de archivo). Así HTK puede saber en qué archivo de audio se encuentra cada palabra.




```
~/proyecto/etc\$ ../scripts/prompts2mlf words.mlf prompts1:40.train
```

5.2 HTK: Master Label File de fonemas

HLEd es un editor que sirve para diversos fines. En este caso, tomará el MLF de palabras y las separará según le indica el diccionario y empleando las ediciones según especifica `mkphones0.led`.




```
HLEd -l '*' -d dictl40 -i phones0.mlf mkphones0.led words.mlf
```

También es necesario generar uno con sp:

```
HLEd -l '*' -d dictl40 -i phones1.mlf mkphones1.led words.mlf
```

`mkphones0.led` Expande los palabras en fonemas, inserta silencios y elimina `sp`:

```
EX
IS sil sil
DE sp
```

`mkphones1.led` no aplica `DE sp`.

Así, se obtiene un Master Label File `phones0.mlf` de fonemas:

```
#!MLF!#
"*/fulano_001.lab"
sil
n
ow
aa
bb
y
ey
n
d
ow
ow
bb
hh
ey
s
y
ow
n
ey
s
aa
s
iy
k
ey
dd
ow
aa
k
```

```
ow
rx
dd
aa
dd
ow
sil
.
"*/fulano_002.lab"
sil
aa
f

...
```

Se observa que el asterisco sirve para que tenga una referencia que se pueda usar en cualquier directorio.

6 Hidden Markov Model (HMM)

En esta sección se procede a:

- ✔ Generar un modelo prototipo de Markov inicial genérico para 1 fonema.
- ✔ Generalizar el prototipo para todos y cada uno de los fonemas.
- ✔ Aplicar el entrenamiento previo a Viterbi.

6.1 Analogía con TP's 6 y 7: Expectation Maximization y Baum-Welch

1.1. TP6: Expectation Maximization (EM)

EM es un tipo de aprendizaje no supervisado utilizado para estimar los parámetros de variables los cuales se desconocen y no se tienen observaciones. Para ello, el algoritmo busca maximizar la probabilidad a posteriori resolviendo el problema de forma estadística.

El algoritmo es altamente dependiente de los parámetros iniciales por lo cual se lo puede inicializar de dos formas:

- **Bootstrap:** Se inicializan los parámetros iniciales utilizando una estimación previa o valores intuitivos.
- **Aleatorio:** Se mezclan todas las muestras de entrenamiento y se calcula la media y varianza de todas las muestras.

El algoritmo EM posee 2 pasos: paso E y paso M.

Paso E (Expectación)

Consiste en calcular las *responsabilidades*. las mismas se representan con γ. **Para mezclas de gaussianas resulta:**

$$\gamma_k(x) \triangleq p(z=k|x) = \frac{p(x|z=k)p(z=k)}{\sum_{l=1}^{K} p(x|z=l)p(z=l)} = \frac{\pi_k \mathcal{N}(x|\mu_k, \sigma_k^2)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(x|\mu_l, \sigma_l^2)} \tag{5}$$

En la expresión precedente se calcula la probabilidad de que la variable aleatoria z se corresponda a la clase k de las K clases disponibles dado un conjunto de observaciones x. Para ello, debe disponerse de la media y la varianza de cada clase.

Paso M (Maximización)

Luego de la expectación, deben recalcularse los parámetros estimados $\boldsymbol{\Theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi})$ y calcular el *log-likelihood*. **Para mezclas de gaussianas:**

$$n_k \triangleq \sum_{i=1}^{n} \gamma_k(x_i), \quad \textit{cantidad de elementos de clase } k \tag{6}$$

$$\mu_k = \frac{1}{n_k} \sum_{i=1}^{n} \gamma_k(x_i) x_i \tag{7}$$

$$\sigma_k^2 = \frac{1}{n_k} \sum_{i=1}^{n} \gamma_k(x_i)(x_i - \mu_k)^2 \tag{8}$$

$$\pi_k = \frac{n_k}{n} \tag{9}$$

Convergencia

El algoritmo debe converger a la distribución $q(\mathbf{Z})$ considerada la verdadera distribución que representa a la clasificación de las muestras. Para ello, debe maximizarse el *log-likelihood* $LL(q, \mathbf{\Theta})$. Se considera que el algoritmo converge cuando $|LL^j(\mathcal{X}, \mathbf{\Theta}^j) - LL^{j+1}(\mathcal{X}, \mathbf{\Theta}^{j+1})| < \delta$, es decir, cuando nuevas iteraciones $(j+1)$ del algoritmo no generan mejora apreciable en el *likelihood*.

El *log-likelihood* se calcula como:

$$LL(\mathcal{X}, \mathbf{\Theta}) = \ln p(\mathcal{X}|\mathbf{\Theta}) = \sum_{i=1}^{n} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \sigma_k^2) \right\} \tag{10}$$

Si se cumple que:

$$|LL^j(\mathcal{X}, \mathbf{\Theta}^j) - LL^{j+1}(\mathcal{X}, \mathbf{\Theta}^{j+1})| < \delta \tag{11}$$

el algoritmo finaliza, de lo contrario se vuelve al **paso E**.

Clasificación

Un posible resultado de la clasificación de vocales (a, o, u) mediante EM se ilustra a continuación:

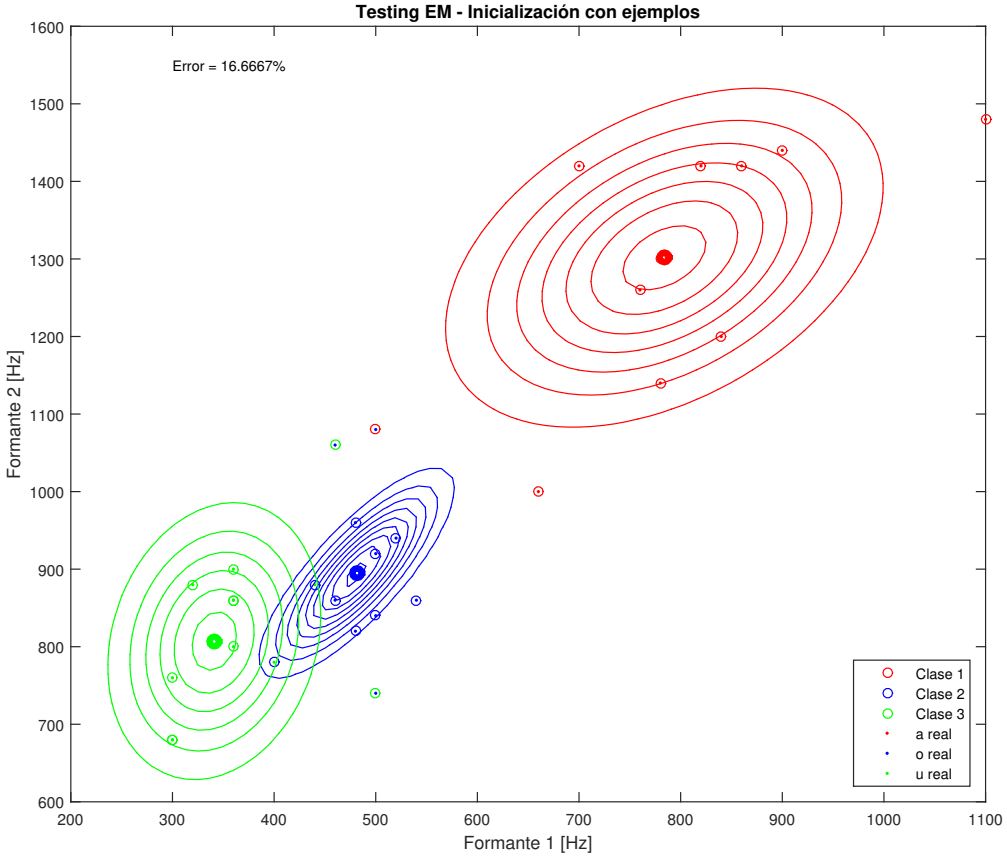


Figura 1: Entrenamiento con EM

Para realizar la clasificación se recurre a la probabilidad a posteriori. Por ejemplo, sean 2 clases $z = 1$ y $z = 2$. Dada una observación $\mathbf{x}$, $\mathbf{x}$ pertenecerá a $z = 1$ si se cumple que:

$$p(z = 1|\mathbf{x}) > p(z = 2|\mathbf{x}) \tag{12}$$

Por la regla de Bayes, se obtiene:

$$\frac{p(\mathbf{x}|z=1)p(z=1)}{p(\mathbf{x})} > \frac{p(\mathbf{x}|z=2)p(z=2)}{p(\mathbf{x})} \tag{13}$$

Esto es,

$$p(\mathbf{x}|z=1)p(z=1) > p(\mathbf{x}|z=2)p(z=2) \tag{14}$$

Ahora bien:

$$p(\mathbf{x}|z=k) = \mathcal{N}(\mathbf{x}|\Theta_k) \tag{15}$$

$$p(z=k) = \pi_k \tag{16}$$

Tomando el logaritmo al producto de 15 y 16 se obtiene el discriminante:

$$g_k(\mathbf{x}) = \log\ p(\mathbf{x}|z=k)p(z=k) \tag{17}$$

$$g_k(\mathbf{x}) = \log\ \mathcal{N}(\mathbf{x}|\Theta_k)\pi_k = -\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k) - \log\left[(2\pi)^n|\Sigma_k|\right] + \log\pi_k \tag{18}$$

Siendo n la cantidad de dimensiones de la muestra (para el ejemplo de las vocales son 2 dimensiones). Por lo tanto, la observación $\mathbf{x}$ pertenecerá a la clase $z = k$ si k maximiza:

$$h(\mathbf{x}) = \underset{z}{arg\,max}\,(g_z(\mathbf{x})) \tag{19}$$

Volviendo al ejemplo original, $\mathbf{x}$ pertenece a la clase 1 si $g_1(\mathbf{x}) > g_2(\mathbf{x})$.

1.2. TP7: Baum-Welch, algoritmo forward-backward

Inicialización

Un modelo de Markov se define como:

$$\theta = \{\{a_{ij}\}, \{b_j(y_t)\}, \{\pi_j\}\} \tag{20}$$

En primer lugar se comienza inicializando la matriz de transiciones:

$$A = \{a_{ij}\} = P(X_t = j|X_{t-1} = i) \tag{21}$$

Que indica la probabilidad de que el estado t-ésimo sea j dado que el $t-1$ fué i. Usualmente se inicializa de manera equiprobable *hacia adelante* con probabilidad nula para estados previos.

Luego se define la probabilidad de que se observe y_i en el estado $X_t = j$:

$$b_j(y_i) = P(Y_t = y_i|X_t = j) \tag{22}$$

Siendo la distribución del estado inicial:

$$\pi_i = P(X_1 = i) \tag{23}$$

Forward

La probabilidad $\alpha_i(t) = P(Y_1 = y_1, ..., Y_t = y_t, X_t = i|\theta)$ de observar $Y_1 = y_1, ..., Y_t = y_t$ en el estado $X_t = i$ es:

$$\alpha_i(1) = \pi_i b_i(y_1) \tag{24}$$

$$\alpha_i(t+1) = b_i(y_{t+1})\sum_{j=1}^{N}\alpha_j(t)a_{ji} \tag{25}$$

Backward

La probabilidad $\beta_i(t) = P(Y_{t+1} = y_{t+1}, ..., Y_T = y_T, X_t = i|\theta)$ de observar $Y_{t+1} = y_{t+1}, ..., Y_T = y_T$ en el estado $X_t = i$ es:

$$\beta_i(T) = 1 \tag{26}$$

$$\beta_i(t) = \sum_{j=1}^{N} \beta_j(t+1) a_{ij} b_j(y_{t+1}) \tag{27}$$

Actualización

La probabilidad de estar en el estado $X_t = i$ dada la observación Y es:

$$\gamma_i(t) = P(X_t = i|Y, \theta) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{j=1}^{N} \alpha_j(t)\beta_j(t)} \tag{28}$$

La probabilidad de estar en el estado $X_t = i$ y que el siguiente sea $X_{t+1} = j$ dada la observación Y es:

$$\varepsilon_{ij}(t) = P(X_t = i, X_{t+1} = j|Y, \theta) = \frac{\alpha_i(t) a_{ij} \beta_j(t+1) b_j(y_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i(t) a_{ij} \beta_j(t+1) b_j(y_{t+1})} \tag{29}$$

Siendo los nuevos parámetros de Markov:

$$\pi_i^{nuevo} = \gamma_i(1) \tag{30}$$

$$a_{ij}^{nuevo} = \frac{\sum_{t=1}^{T-2} \varepsilon_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{31}$$

$$b_i^{nuevo}(r) = \sum_{t=1}^{T} \frac{\mathbb{1}(y_t = r)\gamma_i(t)}{\sum_{t=1}^{T} \gamma_i(t)} \tag{32}$$

6.2 HTK: HMM Inicial

Existen dos formas de generar un HMM. La primera es realizando un **bootstrap** el cual consiste en utilizar datos de medias y varianzas disponibles o estimados previamente. La segunda se le suele llamar **flat start** y en la misma se toma la media y la varianza de la totalidad de los datos, generalmente colocando un piso para la varianza (ver `varFloorN`). En el presente trabajo se aplica el segundo método para lo cual se utiliza la herramienta HCompV.

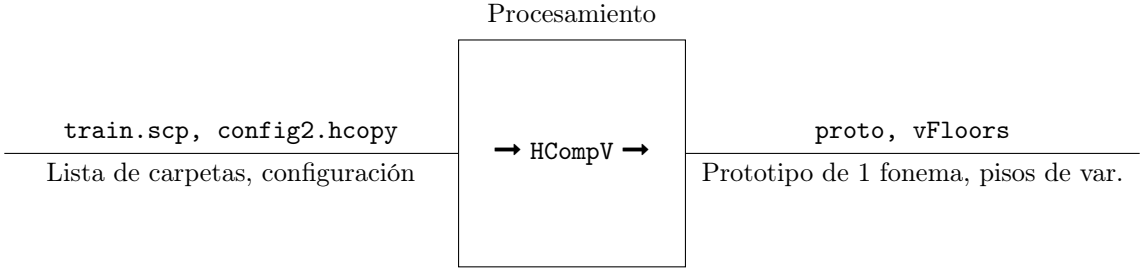
6.3 HTK: Uso de HCompV

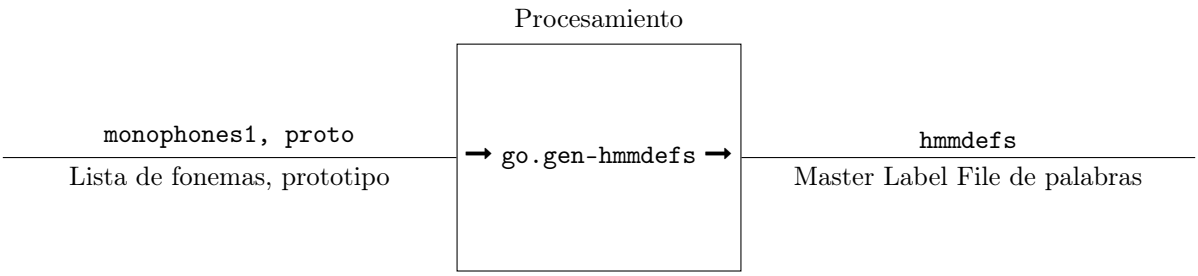
HCompV recibe una lista de MFCC's en un script y un archivo de configuración y devuelve un prototipo `proto` del modelo de uno de los fonemas (con la media y varianza de todos los fonemas) junto con un archivo de pisos de varianzas. Luego, debe generarse el modelo de Markov de la totalidad de los fonemas, todos con la misma varianza y media, para lo cual se utilizará el script `go.gen-hmmdefs` el cual recibe el prototipo y la lista de fonemas y devuelve el **modelo de Markov inicial**.

La herramienta se invoca como:

```
HCompV -C ../config/config2.hcopy -f 0.01 -m -S train.scp -M hmm0 ../modelos/proto
```

Donde -f indica que debe devolver un archivo con los pisos de las varianzas llamado `vFloors`.





La configuración solo contiene información del archivo de salida y le indica que compute los coeficientes δ y de aceleración.

```
TARGETKIND = MFCC_0_D_A
```

`go.gen-hmmdefs` se ejecuta como:

```
~/proyecto/etc$ ../scripts/go.gen-hmmdefs monophones1 ../modelos/proto > hmmdefs
```

El archivo `train.scp` contiene los nombres de las carpetas de train:

```
fulano
mengano
```

Finalmente, el prototipo luce como:

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "proto"
<BeginHMM>
 <NumStates> 5
 <State> 2
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪ 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪  1.0 1.0
 <State> 3
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪ 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪  1.0 1.0
 <State> 4
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪ 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪  1.0 1.0
 <TransP> 5
  0.0 1.0 0.0 0.0 0.0
  0.0 0.6 0.4 0.0 0.0
  0.0 0.0 0.6 0.4 0.0
  0.0 0.0 0.0 0.7 0.3
```

```
  0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

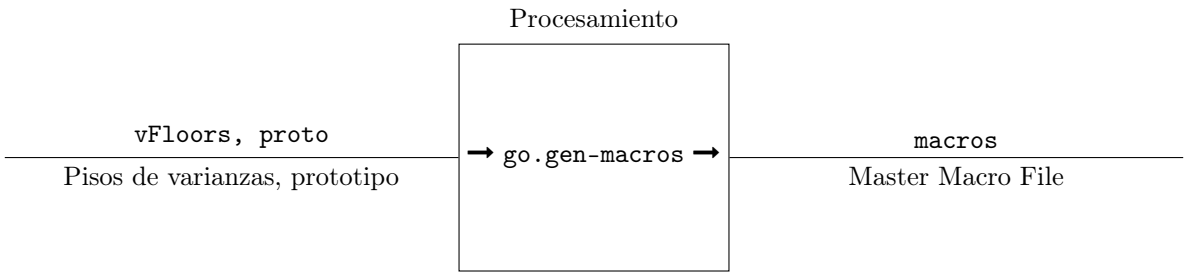
Mientras que `hmmdefs`:

```
~h "aa"
<BeginHMM>
 <NumStates> 5
 <State> 2
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪ 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪  1.0 1.0
 <State> 3
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪ 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪  1.0 1.0
 <State> 4
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39
        1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪ 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
            ↪  1.0 1.0
 <TransP> 5
  0.0 1.0 0.0 0.0 0.0
  0.0 0.6 0.4 0.0 0.0
  0.0 0.0 0.6 0.4 0.0
  0.0 0.0 0.0 0.7 0.3
  0.0 0.0 0.0 0.0 0.0
<EndHMM>
~h "sp"
<BeginHMM>
 <NumStates> 5
 <State> 2
   <Mean> 39
        0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪ 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
            ↪  0.0 0.0
   <Variance> 39

...
```

Debe prestarse especial atención de que el prototipo solo modela un fonema genérico mientras que en `hmmdefs` se modela la totalidad de los fonemas.

Luego se procede a generar los macros `macros` usando el script `go.gen-macros`.



Se ejecuta como:

```
~/proyecto/etc$ ../scripts/go.gen-macros ./hmm0/vFloors ../modelos/proto > macros
```

El archivo `macros` tiene la forma:

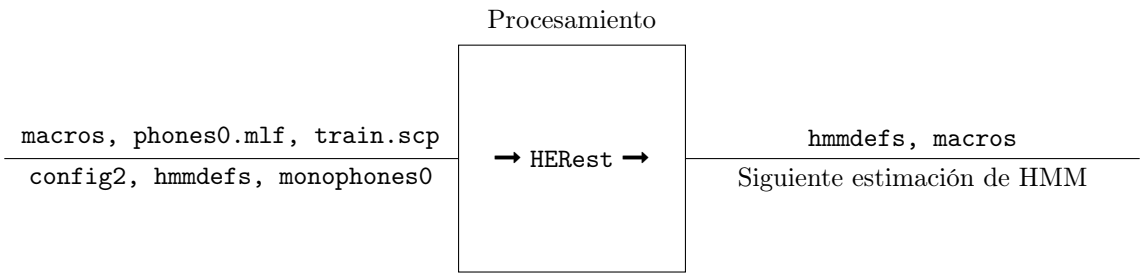
```
~o <VecSize> 39 <MFCC_0_D_A>
~v varFloor1
<Variance> 39
 4.912927e-01 3.201925e-01 4.591970e-01 6.982524e-01 7.528906e-01 6.552078e-01
    ↪ 6.032398e-01 4.897406e-01 4.815772e-01 3.927904e-01 4.544140e-01 3.637407e-01
    ↪ 1.020039e+00 2.270553e-02 1.836995e-02 2.205474e-02 3.440568e-02 3.509371e-02
    ↪ 3.123952e-02 3.142145e-02 3.095316e-02 2.873873e-02 2.637191e-02 2.526928e-02
    ↪ 2.136164e-02 2.365552e-02 3.281180e-03 2.994132e-03 3.485773e-03 5.573777e-03
    ↪ 5.800697e-03 5.454681e-03 5.530963e-03 5.582977e-03 5.185300e-03 4.903159e-03
    ↪ 4.589276e-03 3.917152e-03 3.384365e-03
```

6.4 HTK: Reestimaciones, HERest y HHEd

Los modelos se distribuirán en una carpeta `modelos` de la siguiente forma:

- `Modelos`
- `hmm0` - *modelo inicial con todos los fonemas idénticos* ***(usa monophones1)***
- `hmm1` - *reestimación inicial* ***(usa monophones0)***
- `hmm2` - *reestimación extra según recomienda* `HTK Book` ***(usa monophones0)***
- `hmm3` - *reestimación extra según recomienda* `HTK Book` ***(usa monophones0)***
- `hmm4` - *modificación de* `hmm3` *para incluir el modelo de* `sp`. *No se reestima.*
- `hmm5` - *edición de modelos con* `HHEd`
- `hmm6` - *reestimación* ***(usa monophones1)***
- `hmm7` - *reestimación* ***(usa monophones1)***
- `hmm8` - *reestimación* ***(usa monophones1)***

La función encargada de la reestimación es `HERest`.



Hidden Markov Model 1

`HERest` Se invoca como:

```
HERest -C ../config/config2.hcopy -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H
    ↪  hmm0/macros -H hmm0/hmmdefs -M hmm1 monophones0
```

Este comando reestimará los modelos de `hmm0` en un nuevo modelo `hmm1`.

👍 Siguiendo las recomendaciones del `HTK Book`, se realizan 2 reestimaciones más hasta obtener un `hmm3`.

👍 **Pruning:** `HERest` aplica el algoritmo de *forward-backward* hasta que llega hasta cierto umbral de error. Esto determina la alineación de los modelos acústicos con los estados de la cadena de Markov. Utilizando el método de *pruning* se puede en cierta forma limitar la cantidad de alineamientos con los estados y así reducir la complejidad de cómputo en un orden de magnitud. El *pruning* se establece con la directiva `-t` y establece un umbral inicial de 250, que aumenta de a pasos de 150 en caso de que el error no resulte satisfactorio hasta llegar a 1000 donde se considera que el error se debe a otro tipo de problema *(ver pág. 32 HTK Book)*.

Hidden Markov Models 2 y 3

Repetir lo aplicado en `hmm1` pero reestimando desde 1 a 2 y desde 2 a 3.

Hidden Markov Model 4

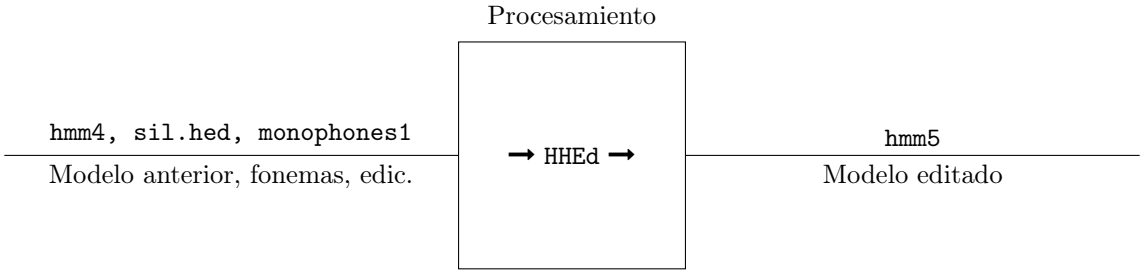
Copiar `hmm3` a `hmm4` y crear un nuevo modelo de `sp` dentro de `hmmdefs` copiando el de sil (dentro de un editor de texto). Notar que en general los fonemas poseen 5 estados, 2 de los cuales son de entrada y salida por lo cual no son estocásticos y no están modelados en `hmmdefs`, por lo cual se encuentran presentes 3 estados en cada fonema. Para generar `sp` solo debe tener 1 de esos 3 estados por lo cual deben ser eliminados 2 estados (estados 2 y 4, por ejemplo). La matriz de transiciones de `sp` tiene que ser de 3x3 eliminando las dos últimas columnas y las 2 anteúltimas filas. Ver que suman 1. *Ver 3.2.2 de HTKBook.*

El modelo de `sp` tiene la forma:

```
~h "sp"
<BEGINHMM>
<NUMSTATES> 3
<STATE> 2
<MEAN> 39
 -1.067701e+01 -4.874971e+00 4.274909e+00 -3.887160e-01 9.980795e-01 2.906510e+00
    ↪ 2.760772e+00 1.870945e+00 2.330526e+00 1.951953e+00 2.819297e+00 3.377707e+00
    ↪ 5.407343e+01 -1.931764e-02 -6.215779e-03 1.894652e-02 1.859230e-02 9.756909e-03
    ↪  1.805758e-02 9.456182e-03 1.322147e-02 1.657445e-02 1.578976e-02 1.411246e-02
    ↪ 1.161560e-02 -3.099222e-02 7.182935e-03 -1.113097e-03 8.243602e-04 -2.897929e
    ↪ -04 -3.069489e-03 -2.106480e-03 -2.271067e-04 -4.022406e-03 -1.378618e-03
    ↪ -8.330615e-03 -3.417627e-03 -4.077938e-03 2.646124e-03
<VARIANCE> 39
 1.602198e+01 1.337535e+01 1.694796e+01 2.974985e+01 3.451905e+01 3.068331e+01
    ↪ 2.752063e+01 2.784737e+01 2.649906e+01 2.505678e+01 2.563358e+01 1.922397e+01
    ↪ 4.382906e+01 8.483797e-01 8.273242e-01 1.043102e+00 1.610910e+00 1.771370e+00
    ↪ 1.752790e+00 1.869733e+00 1.924981e+00 1.877100e+00 1.891734e+00 1.705734e+00
    ↪ 1.437676e+00 1.048591e+00 1.451871e-01 1.497022e-01 1.883451e-01 2.884995e-01
    ↪ 3.294855e-01 3.348276e-01 3.639771e-01 3.726507e-01 3.715966e-01 3.744950e-01
    ↪ 3.369281e-01 2.889684e-01 1.603044e-01
<GCONST> 1.010678e+02
<TRANSP> 3
 0.000000e+00 1.000000e+00 0.000000e+00
 0.000000e+00 9.466090e-01 5.339103e-02
 0.000000e+00 0.000000e+00 0.000000e+00
<ENDHMM>
```

Hidden Markov Model 5

Se genera `hmm5` usando `HHEd` para crear los arcos entre segundo y cuarto estado de `sil` y decirle a `sp` que comparte su estado central con el estado central de `sil`. Notar que `sil.hed` tiene las instrucciones para crear los arcos.



Ejecutar `HHEd` de la siguiente forma:

```
HHEd -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed monophones1
```

`sil.hed` tiene la forma:

```
AT 2 4 0.2 {sil.transP}
AT 4 2 0.2 {sil.transP}
AT 1 3 0.3 {sp.transP}
TI silst {sil.state[3],sp.state[2]}
```

Donde los comandos `AT` indican las conexiones entre estados (de 2 a 4 en sil, de 4 a 2 en `sil` y de 1 a 3 en `sp`) y el comando `TI` conecta el estado central de `sil` con el estado central de `sp`.

Hidden Markov Models 6, 7 y 8

Se ejecuta `HERest` 3 veces más pero con `sp` (consecuentemente hay que usar `monophones1` y `phones1.mlf`):

```
HERest -C ../config/config2.hcopy -I phones1.mlf -t 250.0 150.0 1000.0 -S train.scp -H
    ↪ hmm5/macros -H hmm5/hmmdefs -M hmm6 monophones1
```

>7 Viterbi

En esta sección se procede a:

- ✅ Generar el vocabulario de test
- ✅ Generar probabilidades de n-gramas en formato DARPA
- ✅ Generar la red de palabras (diagrama de Trellis) para aplicar Viterbi
- ✅ Aplicar Viterbi

7.1 Analogía con TP7: Viterbi

Una vez computados los parámetros del `HMM` $\Phi = (A,\ B,\ \pi)$, se procede a decodificar la secuencia óptima. Para ello, se busca la secuencia de estados $S = (s_1, s_2, ..., s_T)$ que maximiza la probabilidad de que la observación $X = (x_1, x_2, ..., x_T)$ haya generado dicha secuencia, dados los parámetros Φ. Es decir, buscamos maximizar $P(S, X|\Phi)$.

El algoritmo de Viterbi resuelve este problema recordando el camino óptimo a estados previos donde:

$$V_t(i) = P(X^t, S^{t-1}, S_t = I|\Phi) \tag{33}$$

Es la probabilidad de la secuencia de estados más probable en el instante t generada por la secuencia X^t en el estado $i - esimo$ del instante t.

Recordar que $b_j(X_t)$ es la probabilidad de observar X_t en el estado j en el instante t.

El algoritmo de Viterbi resulta entonces:

1. **Inicialización**

$$V_1(i) = \pi_i b_i(X_1), \quad 1 \leq i \leq N \tag{34}$$

$$B_1(i) = 0 \tag{35}$$

2. **Inducción**

$$V_t(j) = \max_{1 \leq i \leq N} [V_{t-1}(i) a_{ij}] b_j(X_t), \quad 2 \leq t \leq T; \quad 1 \leq j \leq N \tag{36}$$

$$B_t(j) = \arg \max_{1 \leq i \leq N} [V_{t-1}(i) a_{ij}], \quad 2 \leq t \leq T; \quad 1 \leq j \leq N \tag{37}$$

3. **Finalización parcial**

$$\text{El más probable} = \max_{1 \leq i \leq N} [V_t(i)] \tag{38}$$

$$s_T^* = \arg \max_{1 \leq i \leq N} [B_T(i)] \tag{39}$$

4. **Seguimiento**

$$s_t^* = B_{t+1}(s_{t+1}^*), \quad t = T-1, T-2, ..., 1 \tag{40}$$

$$S^* = (s_1^*, s_2^*, ..., s_T^*), \quad \text{es la mejor secuencia} \tag{41}$$

7.2 Vocabulario

Se genera el vocabulario (necesario para crear la red) el cual se encuentra compuesto por las palabras de test:

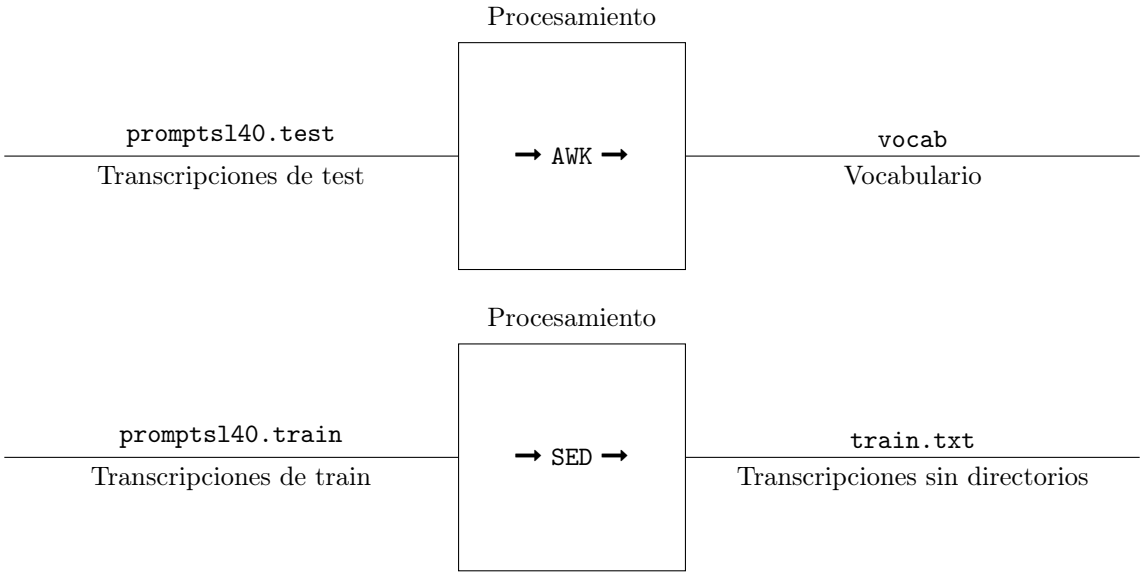
```
~/proyecto/etc$ cat promptsl40.test | awk '{for(i=2;i<=NF;i++){print $i}}'|sort|uniq >
    ↪ vocab
```

Se genera `train.txt` que contiene las transcripciones sin los indicadores de archivos. Para ello se utiliza SED:

```
~/proyecto/etc\$ sed -r 's/[^\t]*\t//' promptsl40.train > train.txt
```

- `-r` usa expresiones extendidas
- `s` indica sustitución
- `[^\t]*` busca cualquier cadena seguida de un tab
- `\t` agrega el tab
- `/` substituye por nada (elimina)
- `/` no indica ningún flag

Funcionamiento: `'s/loquebusco/loquereemplazo/flags'`



`train.txt` tiene la forma:

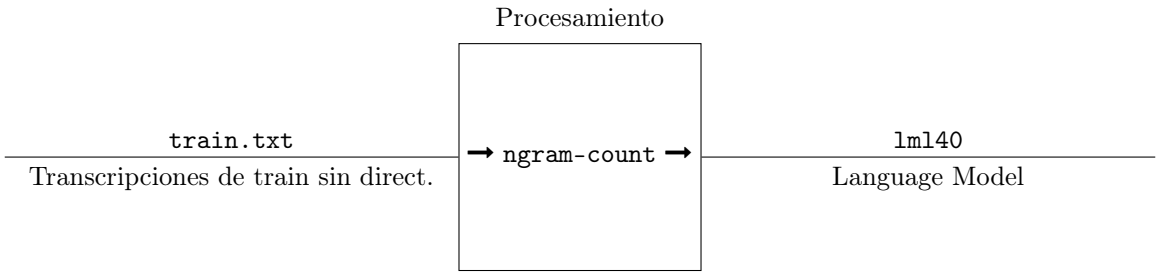
```
no habiendo objeciones así quedó acordado
a fin de año hará el balance de la situación
este programa debe ejecutarse en todos los niveles
no debe interpretarse esto como que uno es más rico
la pareja partió esta madrugada para la capital
a la sazón los locales no disfrutaban de inviolabilidad
la princesa del japón niega que esté embarazada
la esposa y su hija estaban con el cantante en roma
ninguno de los ataques causó daños ni heridos
```

7.3 n-gramas

Se generan los modelos de lenguaje: se genera el archivo `lm140` el cual contiene todas las probabilidades logarítmicas de unigramas y bigramas (formato DARPA). De **no** encontrarse en los archivos de train la palabra `x` precedida por `y`, dicho bigrama será inexistente. Copiar el programa ngram-count en el proyecto y ejecutar:

```
~/proyecto$ ./ngram-count -order 2 -text etc/train.txt -lm -lml40 -ukndiscount2 -vocab
    ↪ etc/vocab
```

👍 `-order 2` *indica que se utilizarán n-gramas de orden hasta n=2 (bigramas).*
👍 `-ukndiscount2` *usa el método original de descuento de Kneser-Ney de orden 2.*



7.4 Trellis, HBuild

Edición del vocabulario

Es necesario que la red modele el comienzo y fin de línea. Se añade `<s>` y `</s>` a `vocab` en un archivo nuevo llamado `vocabs`.

`vocab` y `vocabs` tendrán la forma:

`vocab`

```
a
añadió
abajo
abandonada
...
zona
zonas
zulú
zulema
```

`vocabs`

```
<s>
a
añadió
abajo
abandonada
...
zulú
zulema
</s>
```

Edición del diccionario

Además, se edita `dictl40` en un nuevo archivo `dictl40s` agregándole al comienzo `<s> [] sil` donde `<s>` es el elemento que ve en el diccionario, `[]` le indica a HTK que no escriba el `<s>` en el master label file de salida y `sil` es el modelo fonético. Así mismo, agregar `</s> [] sil` al final.

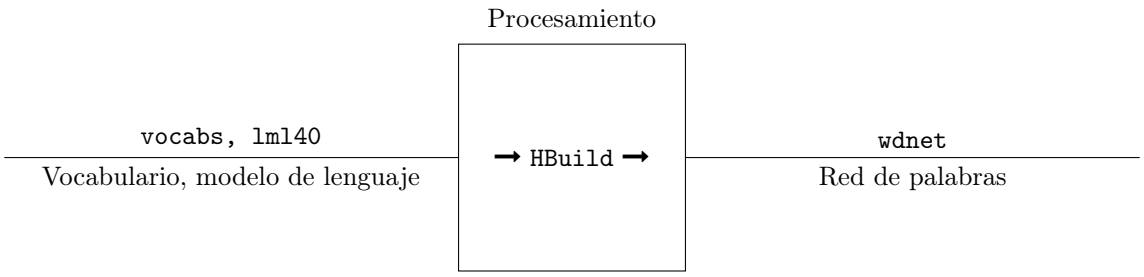
```
<s> [] sil
a             aa sp
abajo         aa bb aa hh ow sp
...
único         uw n iy k ow sp
útil          uw t iy l sp
útiles        uw t iy l ey s sp
</s> [] sil
```

HBuild: creación de la red de palabras

HBuild convierte lml40 de formato DARPA al modelo que necesita HTK con todas las probabilidades entre todas las palabras del modelo entre nodos de la transcripción de training:

```
~/proyecto/etc$ HBuild -s '<s>' '</s>' -n lml40 ./etc/vocabs wdnet
```

👍 `-s` *indica las palabras de entrada y salida del modelo de bigramas.*

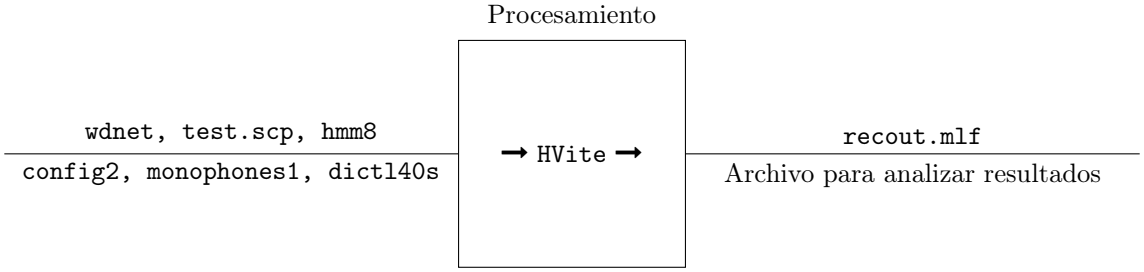


7.5 Viterbi

⌛ Tiempo de procesamiento largo (15min-1h).

Se aplica el algoritmo de Viterbi implementado mediante `HVite`.

```
HVite -C config/config2.hcopy -H modelos/hmm8/macros -H modelos/hmm8/hmmdefs -S etc/
    ↪ test.scp -l '*' -i rec/recout.mlf -w wdnet -p 0.0 -s 5.0 etc/dictl40s etc/
    ↪ monophones1
```



👍 `-p` Cambia la probabilidad de inserción de palabra.
👍 `-s` Multiplica por un factor el *likelihood* ya que algunas probabilidades podrían ser pequeñas.
👍 El archivo `test.scp` contiene la lista de archivos de mfc de test, análogo a `train.scp`.

8 Entrenamiento con mezclas de gaussianas

⌛ **Tiempo de procesamiento largo (15min-2h).**

En la presente sección se profundizará el entrenamiento utilizando mezclas de gaussianas. Esto eleva la complejidad de cálculo por lo cual los algoritmos demoran un tiempo mayor.
Se debe tener en cuenta:

- Las mezclas especificadas deben ser potencia de 2.
- El entrenamiento debe ser progresivo. Es decir, si se desea utilizar mezclas de, por ejemplo, 32 gaussianas, debe aplicarse el procedimiento primero para 2 gaussianas, utilizar los modelos resultantes para 4 gaussianas, luego para 8, luego para 16 y finalmente para 32. No se debe calcular directamente sobre 32 gaussianas.

Los pasos necesarios para realizar y completar el entrenamiento son:

1. Crear/Editar el archivo `cmds.hed` para especificar la cantidad de mezlcas.
2. Ejecutar `HHEd`.
3. Ejecutar `HERest` 2 veces.
4. Repetir los 3 items anteriores hasta obtener la cantidad de mezclas deseadas.
5. Ejecutar `HVite`

-Ejecución HHEd:

```
HHEd -H hmmdefs -H macros -M hmm9 cmds.hed monophones1
```

-Ejecución de HEREst 2 veces (2 directorios):

```
HERest -C ../config/config2.hcopy -I phones1.mlf -t 250.0 150.0 1000.0 -S train.scp -H
    ↪  hmm9/macros -H hmm9/hmmdefs -M hmm10 monophones1
```

Donde `hmm10` es el primer directorio para mezclas de gaussianas y debe ir agregándose `hmm11, hmm12, hmm13`, etc, tomándose de referencia `hmm10, hmm11, hmm12` respectivamente.

-Ejecución HVite:

```
HVite -C config/config2.hcopy -H modelos/hmm11/macros -H modelos/hmm11/hmmdefs -S etc/
    ↪ test.scp -l '*' -i rec/recout2.mlf -w wdnet -p 0.0 -s 5.0 etc/dictl40s etc/
    ↪ monophones1
```

👍 Se puede ejecutar `HVite` luego de ejecutar cualquier `HERest` para verificar que el entrenamiento se comporta como se espera.

El archivo `cmds.hed` tendrá la forma:

```
MU 2 {*.state[2-4].mix}
```

Donde el `2` indica la cantidad de mezclas y `[2-4].mix` indica que los estados que utilizan mezclas son del 2 al 4.

8.1 Referencias de mezclas

Ver página 145 Seccion "10.1 Using HHEd".

9 Resultados

Para analizar los modelos resultantes se ejecuta la función `HResults` de la siguiente forma:

```
HResults -I testref.mlf ../etc/monophones1 recout.mlf
```

Donde `testref.mlf` contiene la traducción *correcta* y `recout.mlf` el resultado de aplicar los modelos.

Resulta conveniente guardar el resultado de `HVite` en distintos archivos para cada cantidad de gaussianas, por ejemplo: `recout1.mlf, recout2.mlf, ..., recout128.mlf`.

Con el procedimiento utilizado se obtuvieron los siguientes resultados para mezclas de hasta 256 gaussianas:

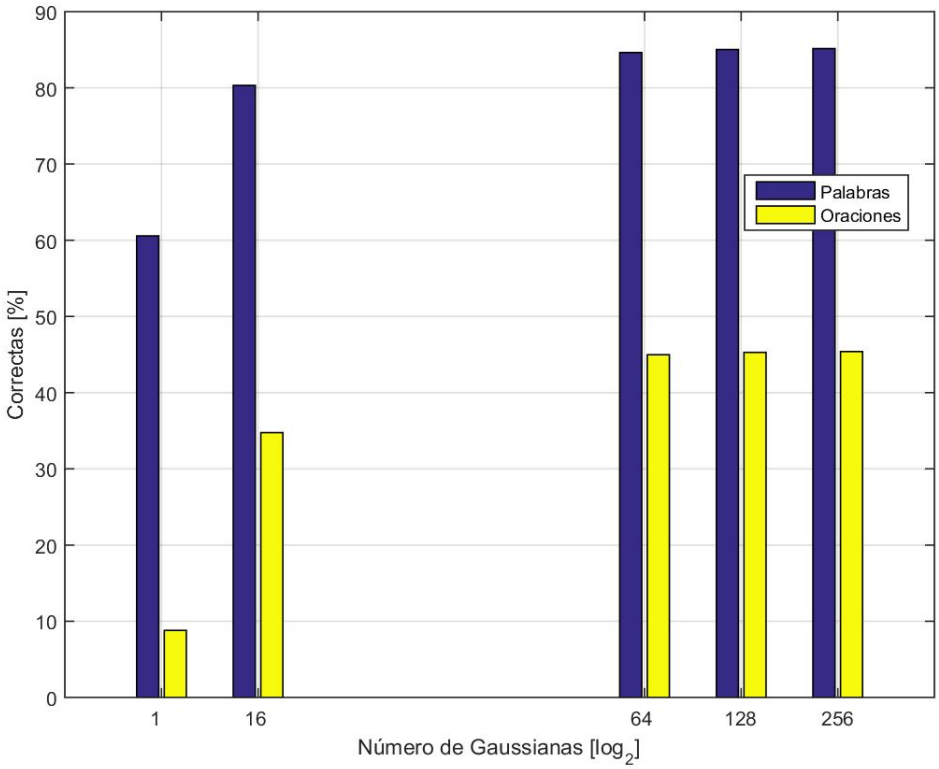


Figura 2: Resultados del proyecto

Conclusiones preliminares

Puede extrapolarse del gráfico precedente que para mezclas mayores a 256 gaussianas el costo computacional que requiere el entrenamiento provocará una mejora difícilmente apreciable que no merece la inversión de tiempo que debe invertirse para ello. Para mejorar la efectividad del algoritmo deben estudiarse nuevas alternativas no contempladas en este proyecto dado que aumentar la cantidad de gaussianas más allá de cierto umbral no provocará un beneficio apreciable.

10 Gramática Finita

En este apartado se procederá a generar un sistema de **gramática finita** el cual, a diferencia de habla continua, posee un marcado comienzo y fin, además de reglas de lenguaje. No se rige por un modelo de lenguaje. El mismo consistirá de un sistema de marcado telefónico mediante el cual se pueda llamar a números o personas. Se utilizarán los modelos ya entrenados de `latino40` por lo cual no resulta necesario un proceso de *training* en esta sección.

10.1 Esquema general

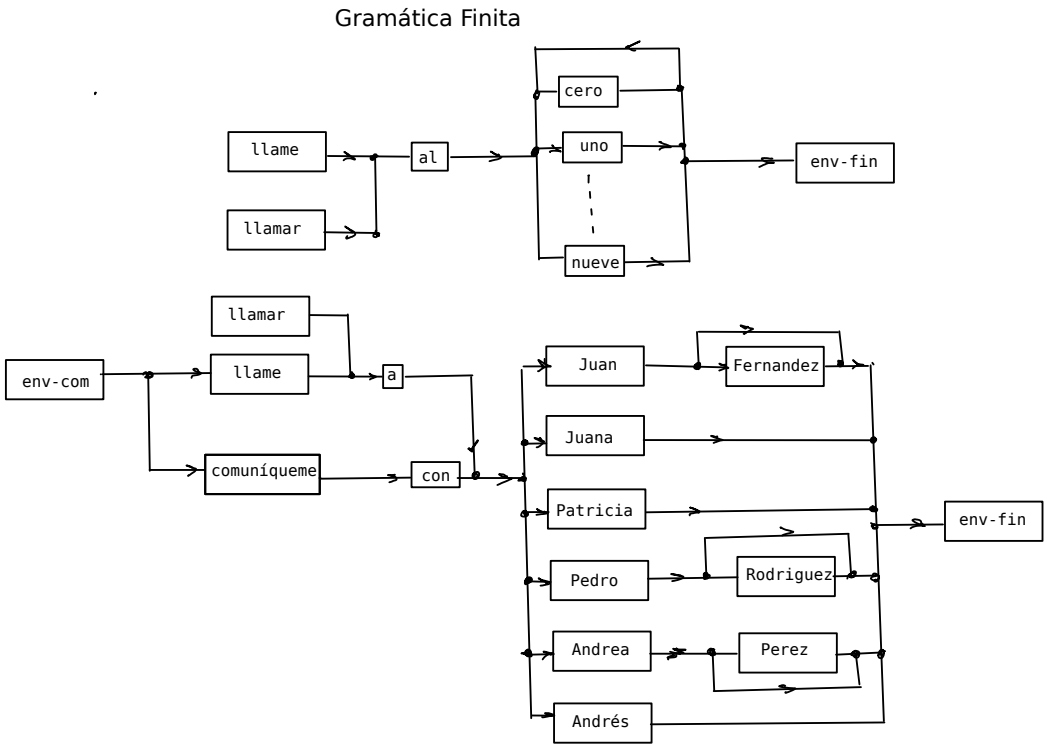


Figura 3: Esquema a implementar de gramática finita

Archivo `grammar`

```
$digit = uno | dos | tres | cuatro | cinco | seis | siete | ocho | nueve | cero;
$name = juan [fernandez] | juana | patricia | pedro [rodriguez] | andrea [perez] |
    ↪ andres;
( enviar-com (( llame | llamar ) al <$digit> | ( comuniqueme con | ( llamar | llame )
    ↪ a ) $name ) enviar-fin )
```

Este archivo impone las reglas de gramática necesarias para generar la red de posibles secuencias de palabras según se ilustra en la figura 3. El carácter `|` es un separador de opciones, `\$` indica una variable, `<>` indica que puede repetirse múltiples veces y `()` marca un elemento en la secuencia.

10.2 Diccionario

Se crea una lista de palabras `wlist.gf` en `/etc` que tendrá la forma:

```
a
andrea
andres
cero
```

```
cinco
comuniqueme
con
cuatro
diego
dos
enviar-com
enviar-fin
fernandez
juan
juana
llamar
llame
nueve
ocho
patricia
pedro
perez
rodriguez
seis
siete
tres
uno
```

Así mismo, se crea un diccionario base `lexicon.gf`:

```
a                                       aa
andrea                                  aa n d rx ey aa
andres                                  aa n d rx ey s
cero                                    s ey rx ow
...
siete                                   s y ey t ey
tres                                    t rx ey s
uno                                     uw n ow
```

Agregando `sp` al diccionario se genera el diccionario `dict.gf`:

```
a                                       aa sp
al                                      aa l
andrea                                  aa n d rx ey aa sp
andres                                  aa n d rx ey s sp
cero                                    s ey rx ow sp
cinco                                   s iy n k ow sp
comuniqueme                             k ow m uw n iy k ey m ey sp
con                                     k ow n sp
cuatro                                  k w aa t rx ow sp
diego                                   d y ey gg ow sp
dos                                     d ow s sp
enviar-com []                           sil
enviar-fin []                           sil
fernandez                               f ey rx n aa n d ey s sp
...
tres                                    t rx ey s sp
uno                                     uw n ow sp
```

Esto resulta análogo a lo que hace global.ded cuando HDMan toma lexicon y genera dict.

10.3 Red de palabras

Para generar la red de palabras (diagrama de Trellis), se genera la red a partir de las reglas de gramática de `grammar`. La implementación se realiza mediante `HParse`:

```
HParse grammar wdnet.gf
```

10.4 Generación de frases aleatorias y grabación

`HSGen` permite generar frases aleatorias a partir de la red generada (la cual a su vez sigue las reglas de `grammar`) a un archivo de texto `promptsgf.test` para luego ser grabadas y usarlas de testing.

```
HSGen -n 200 wdnet.gf dict.gf > promptsgf.test
```

El archivo resulta:

```
1. llamar al seis dos siete dos
2. comuniqueme con andres
3. llame al cuatro cero ocho
4. llame a pedro rodriguez
5. llame a andres
6. llame al tres nueve
...
```

Se graban 200 archivos de audio con el teléfono celular (con buenos filtros de ruido) en `/wav/test` a 16 kHz de frecuencia de muestreo dado que `latino40` fue muestreada a dicha frecuencia.

10.5 Master Label Files

Genero directorios `mfc`

Se edita `go.mfclist` para que genere directorios de test y no de train.

```
if [ $# -ne 1 ]
then
    echo Usar: $0 listatest
    exit 1
fi

ls wav/* >p
cat p |sed 's/wav/mfc/g' >q
paste p q > $1
rm p q
```

Se ejecuta como:

```
./go.mfclist genmfcgf.test
```

Genero master label file

Modifico `prompts2mlf`. La línea:

```
print MLF ("\"$gname.lab\"\n");
```

Se reemplaza por:

```
print MLF ("\"%fname\llab\"\n");
```

Obtengo el archivo `wordsgf.mlf` que contiene los *label file* de palabras ejecutando `prompts2mlf` como:

```
../scripts/prompts2mlf wordsgf.mlf promptsgf.test
```

Genero archivos `mfc`

Se modifica `config.hcopy`, cambiando:

```
SOURCEFORMAT = NIST
```

por:

```
SOURCEFORMAT = WAV
```

Se ejecuta `HCopy` como:

```
HCopy -A -V -T 1 -C ../config/config.hcopy -S ../etc/genmfcgf.test
```

Se genera lista de archivos `mfc`

```
ls datos/mfc/* > testgf.scp
```

10.6 Viterbi

A diferencia de las secciones anteriores, en este proyecto Viterbi converge rápido debido a que la red es mucho más pequeña. **Se utilizan los últimos modelos obtenidos en el training de la primera parte del proyecto.**

Se ejecuta como:

```
HVite -C config/config2.hcopy -H ../modelos/hmm32/macros -H ../modelos/hmm32/hmmdefs -
    ↪ S etc/testgf.scp -l '*' -i recoutgf256 -w etc/wdnet.gf -p 0.0 -s 5.0 etc/dict.
    ↪ gf ../etc/monophones1 &
```

10.7 Resultados

```
HResults -I wordsgf.mlf ../etc/monophones1 recoutgf256
```

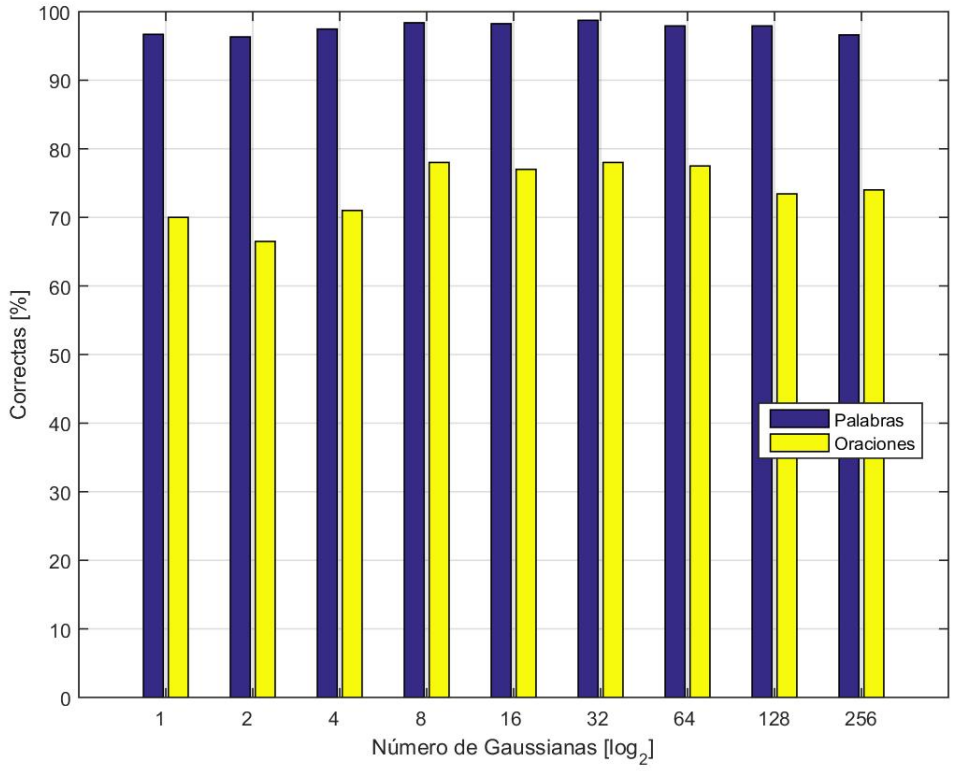


Figura 4: Resultados del proyecto de gramática finita

Se observa que el mejor resultado se obtuvo con 8 gaussianas (muy similar al de 32 gaussianas):

```
====================== HTK Results Analysis =======================
  Date: Sun Sep 30 01:49:15 2018
  Ref : etc/wordsgf.mlf
  Rec : recoutgf8
------------------------ Overall Results --------------------------
SENT: %Correct=78.00 [H=156, S=44, N=200]
WORD: %Corr=98.37, Acc=93.28 [H=1391, D=3, S=20, I=72, N=1414]
===================================================================
```

›11 Conclusiones finales

En procesamiento del habla, específicamente al utilizar HMM, se busca resolver tres principales problemas:

1. **El problema de evaluación.** Dado un modelo Φ y una secuencia de observaciones $X = (X_1, X_2, ..., X_T)$ ¿cuál es la probabilidad $P(X|\Phi)$ de que el modelo genere dichas observaciones? Este problema lo resuelve el algoritmo *Forward-Backward*.

2. **El problema de decodificación.** Dado un modelo Φ y una secuencia de observaciones X ¿Cuál es la secuencia de estados más probable? Este problema lo resuelve el algoritmo de *Viterbi*.

3. **El problema de aprendizaje.** Dado un modelo Φ y una secuencia de observaciones X ¿Cómo podemos ajustar el modelo Φ para maximizar la probabilidad conjunta $\prod_X P(X|\Phi)$? Este problema lo resuelve el algoritmo de *Baum-Welch*.

El uso de HTK para **habla continua** comienza con el análisis espectral de las señales. Las mismas son codificadas a Coeficientes Cepstrum en Escala Mel debido a que se adaptan mejor al sistema auditivo de los humanos. Comparado con los coeficientes LPC, los coeficientes MFCC producen menor error en frecuencia debido a la consideración de la característica logarítmica del oído. Obtenidos los coeficientes MFCC a partir de las muestras de audio, se procede el entrenamiento mediante el algoritmo de *Baum-Welch* (HERest) donde se obtienen los parámetros de una *Cadena Oculta de Markov*. Se realiza una inicialización con HCompV para iniciar a todos los modelos con los mismos parámetros. Luego, se modeliza cada fonema con una distribución estocástica correspondiente a una gaussiana y posteriormente a mezclas de gaussianas. Así, se modeliza la secuencia de fonemas que constituyen el reconocimiento de habla continua, siendo las palabras separadas por estados `sil` los cuales proporcionan flexibilidad al algoritmo dado que delimita automáticamente la separación entre palabras. Finalmente, se utiliza el algoritmo de Viterbi el cual obtiene la secuencia de palabras más probable dados los modelos de HMM obtenidos mediante *Baum-Welch*. A su vez, dicha secuencia de palabras se basa en una red de palabras `wdnet` la cual define las posibles secuencias de palabras y al mismo tiempo la misma fue generada a partir de un diccionario.

Por otro lado, en **gramática finita**, se define la estructura de la red, donde se permite sólo las secuencias de palabras especificadas explícitamente y no cualquier secuencia presente en el training. Se lleva a cabo el testing con el training realizado en habla continua y finaliza el algoritmo.

Finaliza el trabajo con un 95 % de palabras correctas (75 % de oraciones) para el caso de gramática finita y un 85 % de palabras correctas (45 % de oraciones) para el caso de habla continua, habiendo integrado la totalidad de los conocimientos adquiridos a lo largo del curso en el uso básico de HTK.

Referencias

[1] Steve Young, Gunnar Evermann, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev ,Phil Woodlands; *"The HTK Book"*; Microsoft Corporation, Cambridge University Engineering Department. Version 3.2, 2002.

[2] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon; *"Spoken Language Processing"*; Prentice-Hall, 2001.

[3] Frederick Jelinek; *"Statistical Methods for Speech Recognition"*; The MIT Press, 1997.