Proyecto final: Reconocedor de Habla

86.53 Procesamiento del Habla
1° cuatrimestre 2022

Alumno	Padrón	Correo electrónico
Peiretti, Pablo	103592	ppeiretti@fi.uba.ar

Índice

1. Introducción

1.1. Objetivos

El primer objetivo de este proyecto consiste en desarrollar un reconocedor de habla continua en idioma español de 3000 palabras. El segundo, radica en la implementación de otro reconocedor de habla continua pero con la diferencia de consistir de una gramática finita de 30 palabras. Este ultimo tendrá su aplicación como un reconocedor de llamadas telefónicas.

Para lograr llevar a cabo ambas implementaciones, se utiliza el programa *HTK* desarrollado por la universidad de Cambridge. El mismo es un conjunto de herramientas que permite construir modelos ocultos de Markov (*HMM*) y estimar sus parámetros a partir del entrenamiento y sus archivos correspondientes. Por ultimo, con la misma herramienta, se corrobora el resultado de los reconocedores desarrollados con archivos de *test* como se explicara posteriormente.

1.2. Marco teórico

Un reconocedor de habla es un conjunto de modelos estadísticos a través de los cuales se busca estimar el contenido de una señal de habla. Es decir, dada una señal de audio, el reconocedor entrega como salida el conjunto de palabras que los modelos estadísticos encuentran más probable.

En el caso de este proyecto el reconocedor sera de habla continua, es decir, busca identificar grabaciones de habla natural y no de palabras aisladas. Es importante aclarar que las palabras a reconocer deben pertenecer a un vocabulario conocido, no se lograra distinguir palabras fuera del mismo.

En la segunda parte del proyecto se realizara un reconocedor de llamadas telefónicas. Este tipo de sistemas imponen restricciones sobre las posibles secuencias de palabras y se conocen como de gramática finita.

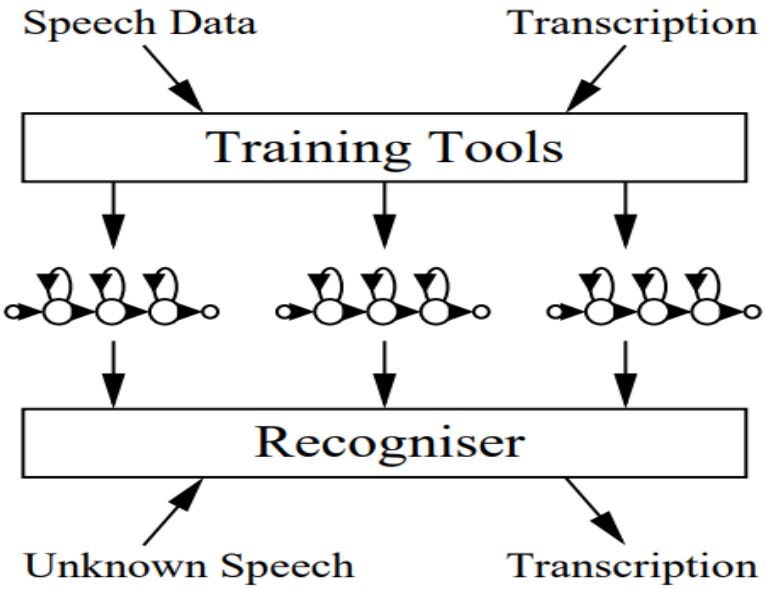

Figura 1.2.1: Esquema conceptual del reconocedor de habla

1.2.1. Modelo de la producción del habla

El modelo de producción de habla mas sencillo consta, en primer lugar, de una señal de entrada generada por el mecanismo de fonación en la laringe. La señal para ruidos vocálicos consiste en un tren de impulsos representando la apertura y cierre de la glotis, y para ruidos consonánticos, la fuente de energía puede ser de ruido turbulento.

Luego, el proceso de modulación se produce en el tracto vocal y en la radiación labial, ambos fenómenos se modelizan como un sistema lineal de M polos dado por H(z).

Además, los ceros debidos a fonemas nasales o al cero de radiación cuando no actúa G(z) (entrada de ruido), son modelizados agregando mas polos al sistema.

Por ultimo, la salida s(n) es la señal de habla modelizada.

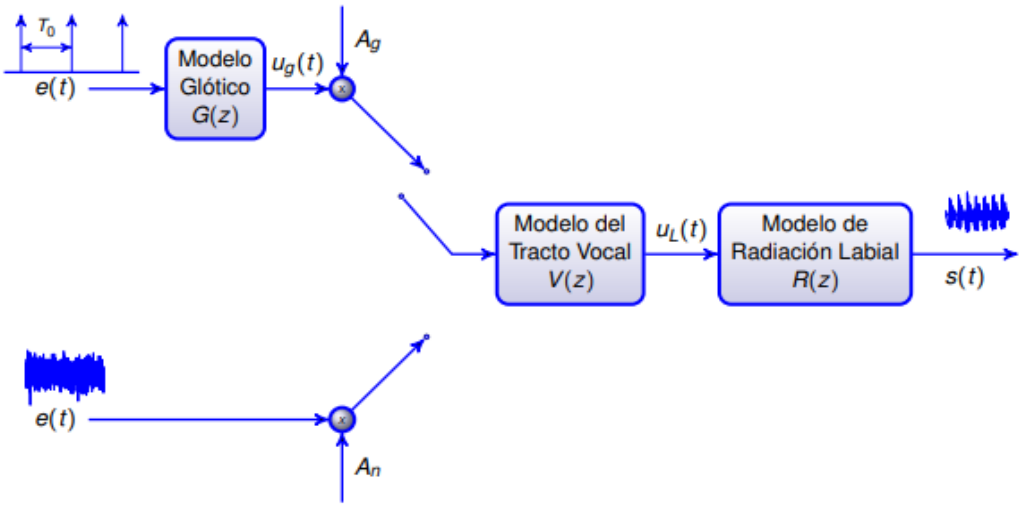

Figura 1.2.1.1: Modelo producción del habla

1.2.2. Coeficientes mel-cepstrum

Con el objetivo de extraer características de las componentes de la señal de audio que sean adecuadas para el reconocimiento, se realizara un procesamiento de las grabaciones que se encuentran en la base de datos comercial *latinos 40*.

En el modelo de habla explicado en la sección anterior, el bloque H(z) sera el que nos brinde información útil para diferenciar a los distintos fonemas, mientras que el tren de impulsos nos dará el tono de voz de cada hablante. En consecuencia, se busca obtener la transferencia de este sistema lineal separándolo de la excitación de entrada.

El proceso de deconvolucionar las señales de forma de obtener únicamente el filtro es muy complejo por lo que se utiliza la técnica de Cepstrum. La misma se basa en aplicarle la transformada de Fourier a la señal, luego el logaritmo y por ultimo, la transformada inversa.

El resultado de este procesamiento se denomina cepstrum y tiene como variable independiente a la quefrencia. Una vez que la señal esta en el dominio de la quefrencia, la parte relacionada con el tracto vocal se concentra en la región de bajas quefrencias y es fácilmente recuperable a partir del filtrado .

Los coeficientes mel-cepstrum son un derivado del procesamiento explicado, donde se agrega información biologica del sistema auditivo para obtener mejores resultados. En primer lugar,

debido a que la escala en la que el oído percibe frecuencias no es lineal, se aplica una escala mel que representa este comportamiento.

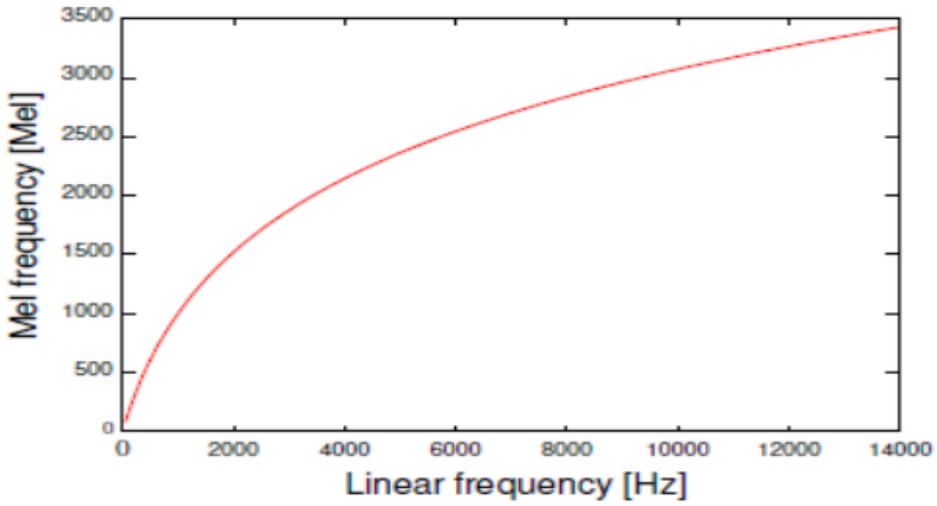

Figura 1.2.2.1: Escala de frecuencias Mel

Luego, como el rango de discriminación de diferentes frecuencias no es igual para distintas frecuencias, se utiliza una serie de filtros triangulares de ancho variable.

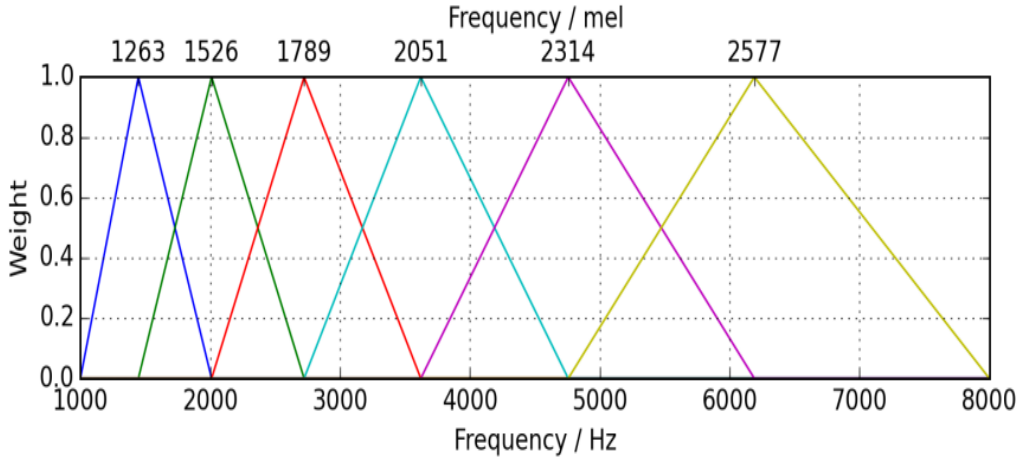

Figura 1.2.2.2: Banco de filtros de bandas criticas

Para obtener los coeficientes se aplicara para cada ventana de señal la DFT, luego se pasara el espectro obtenido por el banco de filtros de bandas criticas y, por ultimo, se obtendrá el modulo del logaritmo de la salida de los filtros y se realizara la transformada de coseno inversa.

Por ultimo, dado que el habla es una señal inherentemente dinámica, se agregaran los coeficientes de velocidad y aceleración que representan dicho aspecto.

1.2.3. Modelos ocultos de Markov

Los modelos ocultos de Markov es el método estadístico que se utilizara en el desarrollo del proyecto ya que resulta muy poderoso para caracterizar datos observados de series estadísticas discretas.

Una cadena de Markov es una secuencia de variables aleatorias que adopta un conjunto discreto de valores llamados estados. Luego, los modelos ocultos, son una extensión de las mismas en la cual se introduce un proceso aleatorio en cada estado que genera símbolos de observación.

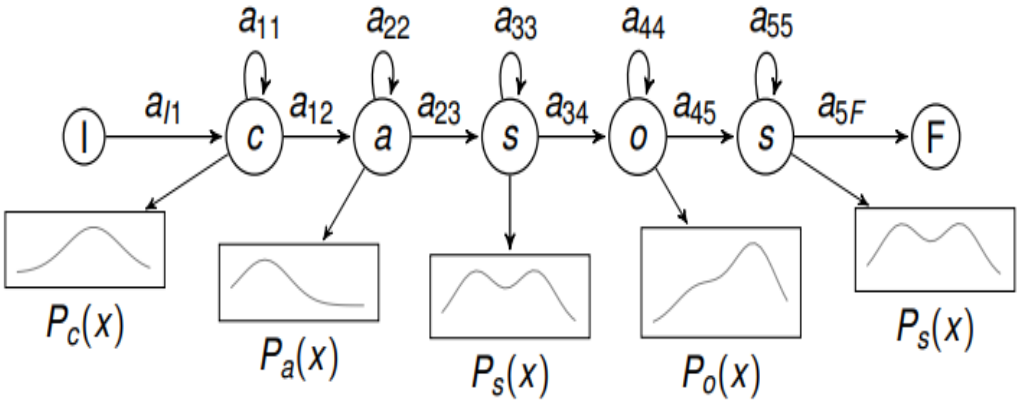

Figura 1.2.3.1: Modelo generativo del habla con HMM

El modelo supone dos hipótesis fundamentales:

- La probabilidad de que una cadena de Markov se encuentre en un estado particular en un determinado instante depende solamente del estado de la cadena en el instante anterior.
- La probabilidad de que una observación x_t sea emitida en el instante t depende solo del estado z_t y es independiente de las observaciones y los estados anteriores.

Los modelos ocultos se caracterizan con una serie de parámetros; En primer lugar se define una topología, en este caso la de Bakis, donde se agrega un estado inicial y otro final no emisores, y se exige que las transiciones solo se pueden dar al mismo estado o al siguiente.

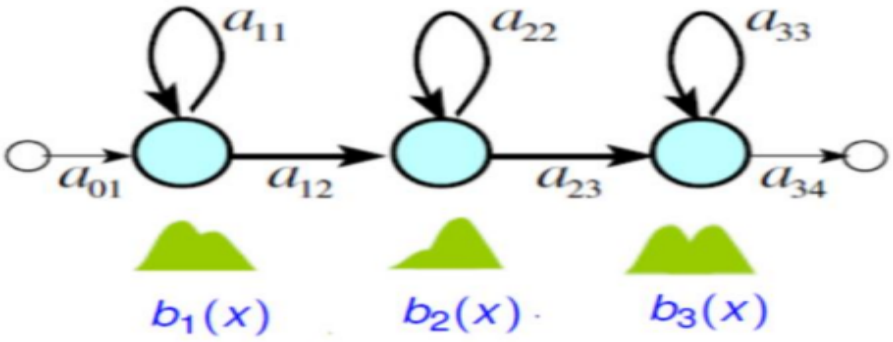

Figura 1.2.3.2: Topologia Bakis

El modelo se completa con una matriz de transición que contiene las probabilidades de transición entre estados (a_{ij}), la distribuciones de observaciones por estado ($b_j(x)$) y la distribución de estado inicial (π_i).

En el reconocedor de habla continua desarrollado se busca resolver dos de los problemas básicos de los HMMs. El primero consiste en estimar los parámetros mencionados del modelo, este paso se conoce como entrenamiento y requiere una secuencia de observaciones de entrenamiento y valores iniciales de los parámetros. Una vez obtenido el modelo resultado del entrenamiento, el otro problema es hallar la secuencia de estados mas probable dada una secuencia de observaciones.

1.2.4. Modelos acústicos

En este proyecto se realizara un entrenamiento de sub-unidades (fonemas) ya que utilizar un modelo de palabras completas traería complicaciones como la poca flexibilidad o la dificultad de encontrar datos suficientes de entrenamiento para cada palabra.

Se empleara una topología de Bakis de tres estados emisores para cada fonema debido a un compromiso entre eficiencia computacional y complejidad del modelo . Por otro lado, se supondrá que las densidades de probabilidad son gaussianas o mezcla gaussianas (a definir la cantidad mediante prueba y error).

Por ultimo, se incorporara un modelo para silencios largos presentes al inicio y fin de las grabaciones (/sil/) y otro para pausas cortas propias del habla leido (/sp/). Este ultimo tiene una topología diferente, posee un solo estado emisor y existe una transición entre el estado inicial y el final ya que el silencio puede ser evadido.

1.2.5. Modelos de lenguaje

Los modelos de lenguaje, en el contexto de reconocimiento de habla, pueden ser descriptos como un método que asigna probabilidades a la ocurrencia de párrafos en un texto. En el desarrollo del proyecto se utilizaran frecuencias relativas para calcular estas probabilidades, en el estado del arte, ya se usan métodos aplicando técnicas de machine learning.

Se emplearan modelos de bi-gramas, es decir, agrupaciones de dos palabras. Sin embargo, puede provocarse en muchos casos que no se encuentren en el texto de entrenamiento ejemplos de ciertos bi-gramas y, por lo tanto, la probabilidad resulte nula. En estos casos se utilizaran técnicas de suavizado, cuyo objetivo es asignar probabilidades mínimas a sucesos que tienen ocurrencia nula.

Existen muchos métodos de suavizados, en el caso de este proyecto, se utilizara el metodo de back-off conocido como *Kneser-Ney* que consiste en la siguiente asignación de probabilidades:

$$P_{KN}(Z|h) == \begin{cases} \frac{N(z,h)-d}{N(h)} & \text{si } N(z,h) > 0, \\ \gamma(h) \cdot P_{KN}(Z|h') & \text{si } N(z,h) = 0, \end{cases}$$

$$P_{KN}(Z|h') = \frac{N_+(\cdot, h', z)}{N_+(\cdot, h', \cdot)} \tag{1.1}$$

Donde N simboliza la cantidad de ocurrencias de un evento, N_+ la cantidad de contextos diferentes donde se produce un evento y d es un valor entre 0 y 1 que se estima empíricamente según el numero de palabras que ocurrieron 1 y 2 veces.

1.2.6. Algoritmo de Baum-Welch

El algoritmo de Baum-Welch es una adaptación del algoritmo EM a los modelos de HMM y, mediante un proceso iterativo, halla los parámetros que maximizan la verosimilitud de las secuencias de observaciones de los datos de entrenamiento. Es decir, busca estimar los parámetros de los modelos HMM.

El algoritmo consta de dos etapas; La primera, conocida como paso E, consiste en calcular la responsabilidades y las probabilidades de transición entre estados. Luego se realiza el paso M que busca obtener los nuevos valores de los parámetros μ_j, Σ_j y a_{ij}. Por ultimo, se calcula el log-likelihood con los nuevos valores de los parametros, se compara con el anterior y se corrobora la convergencia.

Además, este algoritmo como todo algoritmo iterativo, requiere definir valores iniciales de los parámetros y calcular un valor inicial del log-likelihood. En el caso de nuestro proyecto, los valores iniciales para cada fonema seran iguales (exceptuando a /sp/), logrando lo que se conoce como un *flat start*.

1.2.7. Algoritmo de Viterbi

El algoritmo de Viterbi se utiliza con el objetivo de definir, de todas las secuencias que en el instante t se encuentran en el estado j, cual es la mas probable dada una secuencia de observaciones.

Se utiliza el algoritmo de Viterbi para el reconocimiento, generando una matriz de transición que resulta de combinar la probabilidad de transiciones entre los estados de cada palabra y la probabilidad entre palabras. Luego, a partir de la matriz de transición y la probabilidad b_j de cada estado, se encuentra la secuencia de estados óptima.

Finalmente, en el contexto de reconocimiento del habla, se decodifica las palabras a partir de la secuencia de estados.

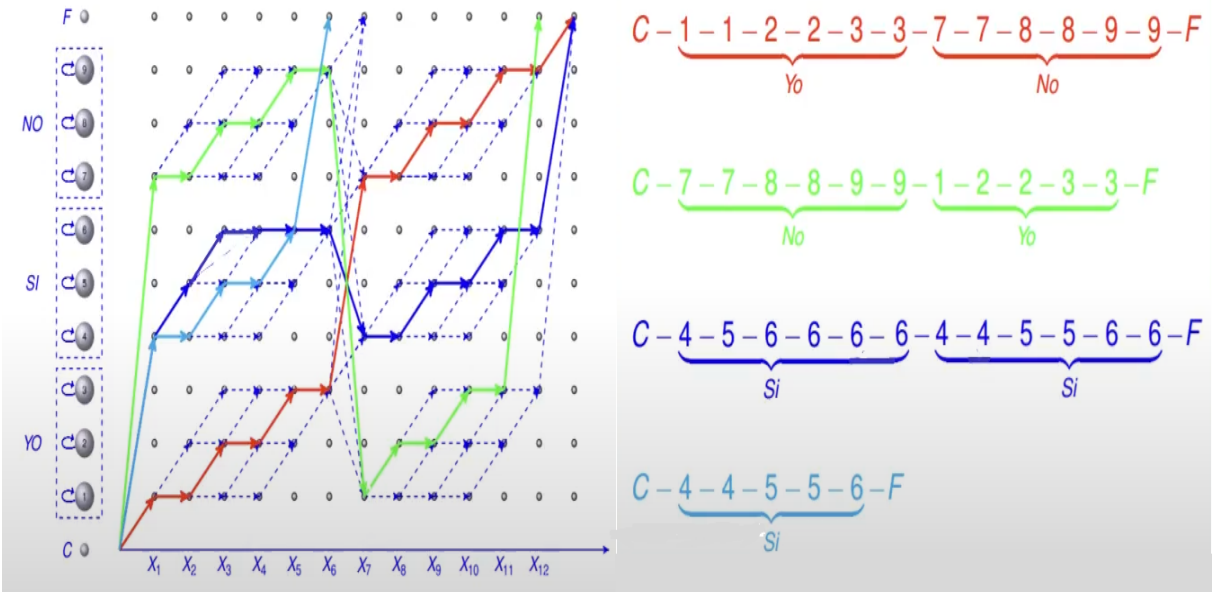

Figura 1.2.7.1: Ejemplo conceptual Viterbi

2. Reconocedor de Habla Continua de 3000 palabras

2.1. Estructura de los Directorios

En primer lugar, se realizo la estructura de directorios requerida para el correcto desarrollo del proyecto con la ayuda de los comandos básicos del interprete *Bash*:

- mkdir: Crea un directorio
- cd: Permite moverse entre directorios del sistema

```
1    46  mkdir proyecto
2    47  cd proyecto/
3    48  mkdir config
4    49  mkdir datos
5    50  mkdir etc
6    51  mkdir lm
7    52  mkdir log
8    53  mkdir modelos
9    54  mkdir rec
10   55  mkdir scripts
11   59  cd datos
12   60  mkdir mfc
13   61  mkdir wav
14   67  cd datos/
15   70  cd mfc
16   71  mkdir test
17   72  mkdir train
```

2.2. Etapa de Parametrización de los datos

El objetivo de esta primer etapa es parametrizar las señales acústicas (archivos wav) obteniendo los coeficientes mel-cepstrum (archivos mfc).

Se comienza obteniendo un conjunto de datos a partir de la base de datos comercial latino 40 con modelos de monofonos. Estos últimos son el tipo de subunidades que se entrenan con la señal correspondiente a la parte del fonema en cuestión sin importar en que contexto ocurran, a diferencia de los trifonos, donde se entrenan solamente si el fonema apareció entre otros dos dados.

La base de datos consiste en emisiones acústicas en formato NIST separadas en los directorios train y devel(test) y, debido a su extensión, se realiza un acceso directo desde el directorio /wav hacia donde se encuentran los archivos.

- ln -s: Crea un acceso directo
- ls: Permite listar los archivos o directorios dentro de un directorio

```
1    75  cd datos/wav
2    78  ln -s /dbase/latino40/train/ train
3    79  ln -s /dbase/latino40/devel/ test
4    81  cd test
5    84  cd af14
6    85  ls
```

Como se menciono con anterioridad, el objetivo de esta etapa es obtener los coeficientes mel-cepstrum, para lo cual, se utiliza la herramienta *hcopy*. La misma requiere un archivo de configuración llamado *config.hcopy* que contiene los parámetros necesarios y, además, se utiliza un archivo ejecutable dado por la catedra, *go.mfclist*, capaz de generar los pares de nombres de archivos .wav - .mfc y sus subdirectorios necesarios.

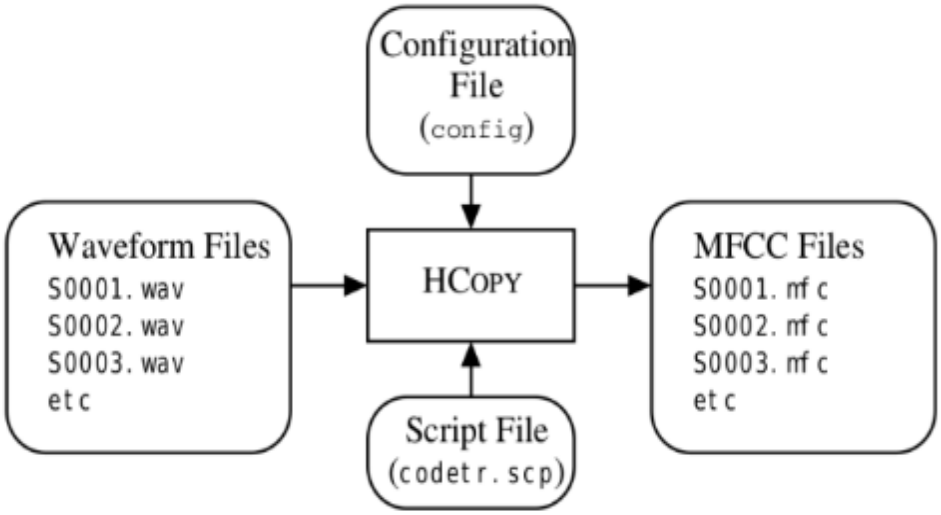

Figura 2.2.1: Estructura de la herramienta *Hcopy*

En primer lugar se importa dentro del directorio /scripts el archivo *go.mfclist*, se modifica sus permisos para que sea ejecutable y se procede a correrlo para generar los archivos genmfc.train y gemfc.test con 3839 y 987 pares respectivamente.

```
1    98 cd scripts
2    99  cp /home/cestien/Documentos/go.mfclist .
3   117  chmod 775 ../scripts/go.mfclist
4   155  ../scripts/go.mfclist genmfc.train genmfc.test
```

Luego, se procede copiando el archivo *config.hcopy* al repositorio local y se analiza cada uno de sus parámetros para un mejor entendimiento.

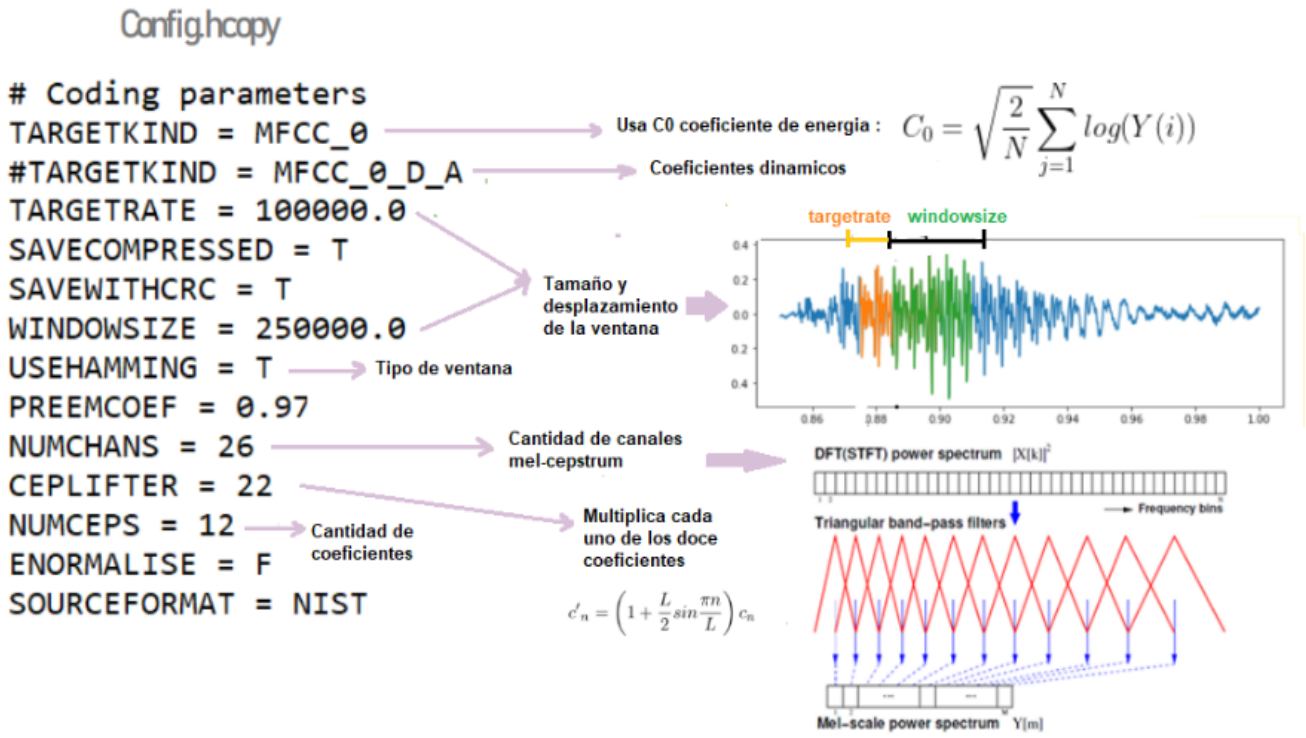

Figura 2.2.2: Parámetros *config.hcopy*

Finalmente, se ejecuta la herramienta *Hcopy* que nos permite generar los archivos .mfc a partir de los .wav.

```
176  HCopy -T 1 -C ../config/config.hcopy -S genmfc.train
177  HCopy -T 1 -C ../config/config.hcopy -S genmfc.test
```

Para visualizar el contenido de los archivos se utiliza el comando Hlist.

```
196  HList -h mfc/test/vm25/vm25_001.mfc
197  HList -F NIST -h -e 0 wav/test/vm25/vm25_001.wav
```

2.3. Etapa de entrenamiento

En esta sección se realizara la etapa de entrenamiento del reconocedor de habla con la herramienta *HERest* que permite realizar un entrenamiento embebido. La misma requiere parámetros iniciales, archivos con coeficientes MFC (generado en la sección anterior), un listado de subunidades en orden de aparición (para cada archivo .wav un listado de todos los fonemas encadenados) y otro de las subunidades a calcular.

2.3.1. Diccionario de pronunciaciones

Con el objetivo de general un diccionario de pronunciaciones, se requiere tener las transcripciones de los archivos .wav generados en la sección anterior a archivos de texto plano. Las mismas se encuentran en los archivos promptsl40.* y, a partir de ellos y con la ayuda

de una herramienta muy poderosa del interprete bash (AWK), se crea una lista de palabras concatenando su contenido, ordenándolo alfabéticamente y eliminando los repetidos.

```
228  cp /home/cestien/Documentos/promptsl40.train
229  cp /home/cestien/Documentos/promptsl40.test

269 cat promptsl40.train promptsl40.test | awk '{for(n=2
; n <= NF ; n++) {print $n}}' | sort | uniq > wlistl40
```

Luego, se busca modelar cada palabra como una secuencia de fonemas que, a su vez, son modelados como modelos ocultos de Markov y de esta forma obtener modelos de palabras con sub-unidades fonéticas.

En primer lugar, se importan los archivos *lexicon** que contienen un diccionario de palabras y sus correspondientes pronunciaciones, en este caso se utiliza únicamente las palabras que se encuentren en el listado wlist40.

```
241  cp /home/cestien/Documentos/lexicon* .
```

Seguido, se copia el archivo *global.ded* que consta de la instrucción "AS sp" que sera de utilidad junto con el comando HDMan para agregar al final de cada descomposición en fonemas un silencio corto(/sp/).

```
243  cp /home/cestien/Documentos/global.ded .
```

Por ultimo, se genera el diccionario de pronunciaciones *dict-l40* con la instrucción *HDMan* a partir de los archivos lexicon.*. La instrucción previamente mencionada tiene varios flags que son de gran importancia para el desarrollo del trabajo;

- -n: Se indica el nombre del archivo donde se generara una lista de todos los fonemas utilizados(cantidad de modelos HMM)
- -l : Se indica la ruta para generar un archivo de logs donde se muestran estadísticas que pueden resultar relevantes, por ejemplo, el numero de ejemplos de cada fonema que se tendra para realizar el entrenamiento.

```
293   HDMan -m -w wlistl40 -g global.ded -n monophones+sil -l
../log/hdman.log dictl40 lexicon  lexicon.gf
```

2.3.2. Archivos de etiquetas

En este inciso se busca generar dos archivos *Master Label Files*, uno que contenga la transcripción a palabras de cada archivo .wav y otro que conste de la transcripción a nivel fonema. Este tipo de archivo sera necesario ya que son archivos de texto con una sintaxis especifica que puede ser comprendida por el programa *HTK*.

Se comienza generando un *Master Label Files* de palabras nombrado como *mlftrain* utilizando el script en lenguaje Perl *prompts2mlf*.

```
343  ../scripts/prompts2mlf mlftrain promptsl40.train
```

Posteriormente, se utilizara la herramienta *HLed* para obtener una transcripción a nivel fonema de cada una de las palabras de mlftrain en el archivo *phones0.mlf* y se le pasara como parámetro el archivo mkphones0.led previamente importado en el directorio /etc que contiene la instrucción IS que permite agregar silencios (/sil/) al principio y final de cada frase , la instrucción DE para eliminar las pausas cortas y, por ultimo, EX que reemplaza cada palabra en mlftrain por su pronunciación correspondiente según el archivo de diccionario dictl40.

```
1  366  HLEd -l '*' -d dictl40 -i phones0.mlf
2  mkphones0.led mlftrain
```

El formato de los archivos de etiquetas generados se muestran a continuación:

```
#!MLF!#
"af01_001.lab"
no
habiendo
objeciones
asi
quedo
acordado
.
```

```
#!MLF!#
"*/af01_001.lab"
sil
n
ow
aa
bb
y
ey
n
d
ow
ow
bb
```

Figura 2.3.2.1: Ejemplo archivo de etiquetas

2.3.3. Modelos de fonemas

Una vez que ya se generaron los archivos de etiquetas, se procederá a crear los modelos de los fonemas. En este caso, para cada fonema se utilizaran 5 estados, de los cuales el primero sera inicial y no se podrá volver a el y del ultimo la probabilidad de salir sera nula. Los otros 3 estados son lo que se conoce como estados emisores y tienen asociados una cierta probabilidad de transición a otro estado o de volver a su mismo estado. En el reconocedor de habla desarrollado, la probabilidad de que se de una observación en un estado se modela con una distribución de mezcla gaussiana. Además, los modelos son inicializados con medias nulas y varianza unitaria para luego ser entrenados con el algoritmo de Baum-Welch.

En primer lugar, se crea un directorio para el primer modelo *HMM0* y se importa el archivo *proto* que contiene una plantilla con la estructura de los estados mencionados anteriormente. Luego se hace lo análogo con el archivo *config* el cual indica que, sumados a los coeficientes MFCC estáticos, se van a calcular los coeficientes de velocidad y aceleración. Además, se genera otro archivo *train.scp* que contiene todos los archivos .mfc que se utilizaran durante el entrenamiento.

Habiendo creado todos los archivos necesarios, se utiliza la herramienta HCompV la cual calcula la media y varianza global para un conjunto de datos y devuelve como resultado un archivo master model file (MMF) con el mismo formato que el archivo de datos generado (proto) pero modificado con los nuevos parámetros. Esta herramienta consta de varios flags que resultan muy utiles:

- -C: Establece la configuración descripta en el archivo
- -f: Configura los valores del piso de varianza, el resultado se da en el archivo *Vfloors*
- -m: Indica que debe calcular medias
- -S: Denota los archivos a ser usados en el entrenamiento
- -M: Se indica el nombre del archivo de salida conformado por una nueva versión del archivo proto con los valores actualizados

```
401  mkdir hmm0
402  cp /home/cestien/Documentos/proto .
407  cp /home/cestien/Documentos/config .
415  ls ../datos/mfc/train/*/* >train.scp
420  HCompV -C ../config/config -f 0.01 -m -S
train.scp -M hmm0 proto
424  cat hmm0/vFloors
```

Se continua copiando los archivos *go.gen-macros* y *go.gen-hmmdefs* al directorio /scripts y se les otorga permisos de ejecución. Estos ejecutables, junto con los archivos monophones+sil, VFloors y proto, permiten generan 2 nuevos archivos:

- hmmdefs: Contiene para cada fonema presente en monophones+sil, los modelos ocultos de markov que son una copia de lo obtenido por HCompV con su nombre cambiado al nombre del fonema correspondiente. Esto permite lo que se conoce como comienzo plano, es decir, todos los fonemas empiezan de la misma manera.
- macros: Se encuentra la información del piso de varianza.

```

433  cp /home/cestien/Documentos/go.gen-macros .
434  cp /home/cestien/Documentos/go.gen-hmmdefs ./
435  chmod +x go.gen-macros
436  chmod +x go.gen-hmmdefs
442  cd modelos/
443  ../scripts/go.gen-hmmdefs ../etc/monophones+sil
hmm0/proto  > hmm0/hmmfeds
445  cat ../scripts/go.gen-macros
447  ../scripts/go.gen-macros hmm0/vFloors hmm0/proto
> hmm0/macros
```

Una vez creados los modelos iniciales, estos se re-calculan usando la herramienta *HERest*. Esta herramienta estima los modelos fonéticos usando solo la información de los fonemas que se encuentran en cada transcripción y su orden, sin conocer su ubicación exacta en la frase.

La misma requiere como entrada la lista de mfc a utilizar como entrenamiento, las transcripciones a nivel fonema obtenidas con HLed y el modelo inicial. Los modelos re-estimados son almacenados en un nuevo MMF. Además, posee los parámetros que se explicaran a continuación:

- -c: Indica el archivo de configuración
- -I: Denota la transcripción a nivel fonema
- -t: Permite podar aquellos términos de las sumas en el algoritmo de forward-backward cuyo likelihood esta debajo de un umbral. Dicho umbral se va incrementando si la re-estimación da un error hasta un limite superior especificado en la opción.
- -s: Archivos a utilizar en el entrenamiento
- -H: Se indica el modelo inicial
- -M: Directorio donde se debe guardar el modelo re-estimado

```
1  448  mkdir hmm1
2  457  HERest -T 1 -C ../config/config -I ../etc/phones0.mlf -t
3  250.0 150.0 1000.0 -S train.scp -H hmm0/macros -H
4  hmm0/hmmfeds -M hmm1 ../etc/monophones+sil
5  458  mkdir hmm2
6  459  HERest  T  1 -C ../config/config -I ../etc/phones0.mlf -t
7  250.0 150.0 1000.0 -S train.scp  -H hmm1/macros -H
8  hmm1/hmmfeds -M hmm2 ../etc/monophones+sil
9  460  HERest -T 1 -C ../config/config -I ../etc/phones0.mlf -t
10 250.0 150.0 1000.0 -S train.scp -H hmm1/macros -H
11 hmm1/hmmfeds -M hmm2 ../etc/monophones+sil
12 461  mkdir hmm3
13 462  HERest -T 1 -C ../config/config -I ../etc/phones0.mlf -t
14 250.0 150.0 1000.0 -S train.scp -H hmm2/macros -H
15 hmm2/hmmfeds -M hmm3 ../etc/monophones+sil
```

2.3.4. Modelos de fonema SP

Una vez que se tiene el modelo re-estimado, se completara la topología del modelo /sil/ permitiendo volver a comenzar y se agregara el modelo /sp/. Este ultimo es necesario ya que, en general, los silencios /sil/ representan el comienzo y el final de una frase y tienden a ser mucho mas largos que el resto de los fonemas y, debido a que las grabaciones son de habla leída y contienen pausas cortas, se debe entrenar otro modelo de pausa corta.

El modelo /sp/ tiene un solo estado emisor y existe una transición entre el estado inicial y final ya que este silencio puede no ocurrir. Para el estado emisor, se inicializara con los valores del estado central del silencio /sil/ ya que ambos tratan de un silencio.

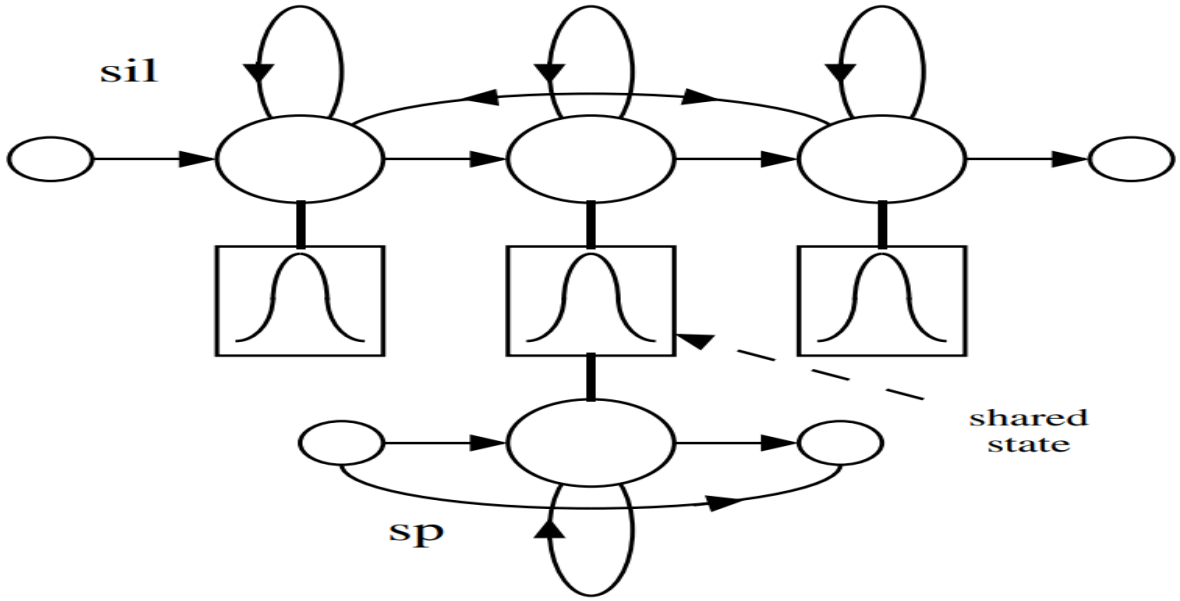

Figura 2.3.4.1: Diagrama conceptual de /sp/ y /sil/

Se comienza copiando los archivos del directorio hmm3 en un nuevo directorio, hmm4. Luego, se edita de forma manual el contenido de hmmdefs, se duplica el modelo de /sil/, se le borra los estados 2 y 4, se eliminan las filas 3 y 4 y columnas 4 y 5 de la matriz de transición y se cambia el nombre por el de /sp/.

A continuación, se importa el archivo *sil.hed* el cual contiene las instrucciones de *add transition* entre los estados 2 y 4, 4 y 2 de /sil/ y, por ultimo, una transición entre los estados 1 y 3 de /sp/. Además, contiene una instrucción para enlazar el estado central de /sil/ con el de /sp/.

```
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]}
```

Figura 2.3.4.2: Archivo sil.hed

Se utiliza nuevamente la herramienta HHEd para modificar el modelo HMM en base al archivo sil.hed, agregando las transiciones y enlazando los estados indicados. Es necesario, pasarle a HHEd un nuevo archivo *monophones+sil+sp* que contenga a /sp/ para entrenarlo. Con el objetivo de generar este ultimo, se copia el contenido del archivo anterior y se agrega al final el /sp/.

Luego, se duplica el contenido del archivo mkphones0.led en un nuevo *mkphones1.led*, eliminando la instrucción "DE sp", que borraba los sp, y se agrega la instrucción "ME sil sp sil" con la finalidad de mezclar y que la ultima palabra no tenga siempre un /sp/ y un /sil/.

```
1   463   mkdir hmm4
```

```
  464  cp hmm3/* hmm4/
  465  sftp ppeiretti@habla.fi.uba.ar
  486  cp /home/cestien/Documentos/sil.hed .
  490  nano monophones+sil
  492  diff -y monophones+sil monophones+sil+sp
  497  mkdir hmm5
  586  HHEd -H hmm4/macros -H hmm4/hmmdefs -M hmm5 sil.hed
  ../etc/monophones+sil+sp
  593  nano mkphones0.led
```

Posteriormente, y de forma analoga a lo realizado con anterioridad, se utilizara la instrucción HLed para obtener la transcripcion a nivel fonema de cada una de las palabras de mlftrain pero esta vez sin eliminar los /sp/ al final de cada palabra.

Por ultimo, se generan 2 estimaciones con la herramienta HERest de la misma forma que se hizo anteriormente.

```
  595  HLEd -l '*' -d dictl40 -i phones1.mlf mkphones1.led
  mlftrain
  596  diff -y phones0.mlf phones1.mlf | less
  599  mkdir hmm6
  600  mkdir hmm7
  601  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf
  -t 250.0 150.0 1000.0 -S train.scp -H hmm5/macros
  -H hmm5/hmmdefs -M hmm6 ../etc/monophones+sil+sp
  602  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf
  -t 250.0 150.0 1000.0 -S train.scp -H hmm6/macros
  -H hmm6/hmmdefs -M hmm7 ../etc/monophones+sil+sp
```

2.4. Modelo de lenguaje

En este punto del proyecto, ya se realizo la etapa de entrenamiento del modelo acústico, donde se inicio con archivos .wav se los paso a archivos .mfc con la herramienta HCopy, luego se inicializaron los modelos con HCompV y se realizaron estimaciones con HERest hasta llegar al modelo hmm7 que contiene modelos entrenados para cada uno de los fonemas.

El paso siguiente entonces, es generar un modelo de lenguaje, recordando que el mismo es una recopilación de información estadística asociada con un vocabulario específico que ayuda a predecir las palabras que tienen más probabilidades de utilizarse en el dictado de un usuario.

Se implementa un modelo de bigramas (grupo de dos palabras) usando la técnica de conteo de palabras con la herramienta *ngram-count*. La misma utiliza un método conocido como Kneser-Ney discounting que permite suavizar la probabilidad de ocurrencia de bigramas para eventos raros.

La herramienta necesita un vocabulario, en este caso, con todas las palabras de la base de datos test ya que buscamos generar un trellis de estas palabras y si se hiciese con la de train podría llegar a haber palabras que estén en esa y no en la buscada. Para generarlo se hace un procedimiento análogo al de cuando se creo wlistl40:

```
  607  cat ../etc/promptsl40.test | awk '{for(n = 2; n<=NF; n++)
```

```
2   {print $n}}' | sort | uniq > vocab
```

Luego, se agregan al diccionario el inicio y fin de frase para calcular también los bigramas con el inicio y fin:

```
1   610  echo "<s>" >> vocab
2   611  echo "</s>" >> vocab
```

Ademas, ngram-cout, requiere un archivo de texto con frases para entrenar el lenguaje. Es importante que estas frases sean de un estilo similar (no igual) al de las que se busca reconocer para obtener mejores resultados.

```
1   612  awk '{{for(n = 2;n<=NF;n++){printf "%s ",$n}}
2   printf "\n"}' ../etc/promptsl40.train  >train.txt
```

A continuación, se ejecuta la herramienta ngram-count que genera el archivo *lml40* que contiene los modelos de los bigramas de las palabras del vocabulario generado. Es importante remarcar que con el flag *ukndiscount2* se indica que se usa el método de suavizado Kneser-Ney discounting mencionado anteriormente.

```
1   669  /usr/local/speechapp/srilm/bin/i686-m64/ngram-count
2   -order 2 -text train.txt -lm lml40 -ukndiscount2
3   -vocab vocab
```

Para completar el diccionario se le agrega las palabras de inicio y final $< s >$ y $< /s >$.

```
1  1086  echo "<s> [] sil" >> ../etc/dictl40
2  1087  echo "</s> [] sil" >> ../etc/dictl40
```

Por ultimo, se utiliza la herramienta *HBuild* para transformar el archivo lml40 que contiene el modelo de lenguaje al formato lattice. Este formato es una forma de representar el modelo de lenguaje donde se pueden observar los nodos, los arcos y sus probabilidades con una sintaxis tal que HTK los puede reconocer. Se utiliza el flag -s para indicarle las palabras de inicio y fin de frase.

```
1  1093  HBuild -n lml40 -s '<s>' '</s>' vocab wdnet
```

2.5. Etapa de Verificación

En este inciso se busca realizar el reconocimiento y verificar los resultados obtenidos con los modelos desarrollados en las secciones anteriores. Se comienza generando un archivo "test.scp" el cual contiene el nombre de todos los archivos de test.

Posteriormente, para realizar el reconocimiento, se emplea la instrucción *HVite* la cual realiza el algoritmo de viterbi. Se comienza aplicándola al modelo HMM7 generado que consiste en un modelo de únicamente una gaussiana.

La herramienta, como se explico anteriormente, genera una matriz de transición y a partir de ella y de la probabilidad b_j de cada estado, encuentra la secuencia de estados óptima. Finalmente, decodifica las palabras a partir de la secuencia de estados. Se procede explicando cada uno de sus flags:

- -c: Indica el archivo de configuración
- -H: Modelo que se usa para realizar el reconocimiento
- -S: Archivo con el nombre de los archivos de test
- -i: Indica el archivo en formato MLF de salida donde se guarda el resultado del reconocimiento
- -w: Denota el archivo con la probabilidad de transición entre palabras
- -p: Configura la probabilidad de insertar una palabra en falso
- -s: Factor de escala gramatical

```
1146  HVite -C ../config/config -H ../modelos/hmm7/macros
-H ../modelos/hmm7/hmmdefs -S test.scp -l '*' -i recout.mlf
-w ../lm/wdnet -p 0.0 -s 5.0 ../etc/dictl40
../etc/monophones+sil+sp &
```

Con el objetivo de comparar los resultados, se crea un archivo "testref.mlf" donde se encuentra la traducción a nivel texto del contenido del los archivos de test.

```
1155  ../scripts/prompts2mlf testref.mlf ../etc/promptsl40.test
```

Por ultimo, se utiliza la herramienta de analisis de resultados brindada por HTK llamada *HResult*. La misma compara los dos archivos .mlf creados con anterioridad y da métricas de los resultados.

```
1169  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout.mlf
```

Los valores que da como resultado esta herramienta son:

- Sent %Correct: Se refiere al porcentaje de oraciones correctas ($\frac{H}{N}$)
- Word %Correct: Indica el porcentaje de palabras correctas ($\frac{H}{N}$)
- Word Acc: Indica el porcentaje de precisión, tomando en cuenta los errores de inserción ($\frac{H-I}{N}$)

Los resultados obtenidos se encuentran a continuación, donde se puede observar que para el modelo de una sola gaussiana son inaceptables en el contexto de reconocimiento del habla. Se tiene aproximadamente un 60 % de palabras correctas y un 7,5 % de oraciones.

```
---------------------- Overall Results ----------------------
SENT: %Correct=7.50 [H=74, S=913, N=987]
WORD: %Corr=59.95, Acc=47.07 [H=4742, D=415, S=2753, I=1019, N=791
------------------------------------------------------------
```

Figura 2.5.1: Resultados modelo de una gaussiana

2.6. Etapa de refinamiento

Los modelos entrenados constan de una distribución gaussiana para cada estado, con el objetivo de mejorar la estimación, se buscara distribuciones que correspondan con mezclas de gaussianas. La cantidad de gaussianas a utilizar debe ser calculada de forma empírica ya que si se cuenta con muchas se puede dar el problema de overfitting, es decir, si se quiere entrenar un modelo de muchas gaussianas y se tienen pocos datos, el modelo no mejorara con respecto a uno de pocas gaussianas y tendrá un costo computacional mucho mayor.

Una aclaración importante es que al pasar de tener una gaussiana a una mezcla, γ pasara a ser tridimensional ya que ahora, para cada estado, se tendrá una suma de gaussianas en la que se podría estar emitiendo. Además los cálculos del modelo serán ligeramente diferentes pero la herramienta HTK viene preparada para realizarlos de forma correcta.

El proceso para determinar el numero de gaussianas que mejor se adapta a nuestro caso sera ir aumentando el valor en potencias de 2 hasta notar que el porcentaje de reconocimiento deja de mejorar.

El primer paso, con el fin de crear el primer modelo con dos gaussianas, sera volver a crear los directorios necesarios.

```
1181  mkdir hmm2g.0
1182  mkdir hmm2g.1/
1183  mkdir hmm2g.2/
```

Luego se establecen los valores iniciales para los modelos, para lo cual, se utiliza nuevamente la instrucción *HHEd* partiendo desde el ultimo modelo *HMM7*. Esta vez se le agrega el archivo importando desde el directorio de Claudio, *editf2g*, cuyo contenido es la instrucción "Mu m" que aumenta el numero de componentes de la mezcla no difuntos para cada función de densidad de probabilidad.

```
695  cp /home/cestien/Documentos/editf2g ./
1184  HHEd -C ../config/config -T 1 -H hmm7/macros -H
hmm7/hmmdefs -M hmm2g.0 editf2g  ../etc/monophones+sil+sp
```

A continuación, se realiza la estimación de forma análoga a como se hizo con el modelo de una gaussiana, ejecutando el algoritmo de Viterbi y verificando los resultados.

```
1185  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf
-t 250.0 150.0 1000.0 -S train.scp -H hmm2g.0/macros -H
hmm2g.0/hmmdefs -M hmm2g.1 ../etc/monophones+sil+sp
1186  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf
-t 250.0 150.0 1000.0 -S train.scp -H hmm2g.1/macros -H
hmm2g.1/hmmdefs -M hmm2g.2 ../etc/monophones+sil+sp
1188  HVite -C ../config/config -H ../modelos/hmm2g.2/macros
-H ../modelos/hmm2g.2/hmmdefs -S test.scp -l '*' -i recout2g.mlf
-w ../lm/wdnet -p 0.0 -s 5.0 ../etc/dictl40
../etc/monophones+sil+sp &
1190  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout2g.mlf
```

```
---------------------- Overall Results -----------------------
SENT: %Correct=11.75 [H=116, S=871, N=987]
WORD: %Corr=64.64, Acc=53.43 [H=5113, D=370, S=2427, I=887, N=7910]
================================================================
```

Figura 2.6.1: Resultados modelo de dos gaussianas

Finalmente, se repite el procedimiento hasta lograr llegar a alguna conclusión acerca del número de gaussianas mas adecuado para utilizar en este proyecto.

Modelo de 4 gaussianas

```
1195  mkdir hmm4g.0
1196  mkdir hmm4g.1
1197  mkdir hmm4g.2
1198  HHEd -C ../config/config -T 1 -H hmm2g.2/macros
-H hmm2g.2/hmmdefs -M hmm4g.0 editf4g ../etc/monophones+sil+sp
1200  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm4g.0/macros -H
hmm4g.0/hmmdefs -M hmm4g.1 ../etc/monophones+sil+sp
1201  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm4g.1/macros -H
hmm4g.1/hmmdefs -M hmm4g.2 ../etc/monophones+sil+sp
1214  HVite -C ../config/config -H ../modelos/hmm4g.2/macros
-H ../modelos/hmm4g.2/hmmdefs -S test.scp -l '*' -i
recout4g.mlf -w ../lm/wdnet -p 0.0 -s 5.0
../etc/dictl40 ../etc/monophones+sil+sp &
1215  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout4g.mlf
```

```
---------------------- Overall Results -----------------------
SENT: %Correct=20.77 [H=205, S=782, N=987]
WORD: %Corr=71.59, Acc=63.30 [H=5663, D=323, S=1924, I=656, N=7910]
================================================================
```

Figura 2.6.2: Resultados modelo de cuatro gaussianas

Modelo de 8 gaussianas

```
1219  mkdir hmm8g.0
1220  mkdir hmm8g.1
1221  mkdir hmm8g.2
1225  nano editf4g
1226  HHEd -C ../config/config -T 1 -H hmm4g.2/macros
```

```
-H hmm4g.2/hmmdefs -M hmm8g.0 editf8g ../etc/monophones+sil+sp
1227  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm8g.0/macros -H
hmm8g.0/hmmdefs -M hmm8g.1 ../etc/monophones+sil+sp
1228  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm8g.1/macros -H
hmm8g.1/hmmdefs -M hmm8g.2 ../etc/monophones+sil+sp
1230  HVite -C ../config/config -H ../modelos/hmm8g.2/macros
-H ../modelos/hmm8g.2/hmmdefs -S test.scp -l '*' -i
recout8g.mlf -w ../lm/wdnet -p 0.0 -s 5.0 ../etc/dictl40
../etc/monophones+sil+sp &
1232  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout8g.mlf
```

```
---------------------- Overall Results -------------------------
SENT: %Correct=29.58 [H=292, S=695, N=987]
WORD: %Corr=77.55, Acc=70.96 [H=6134, D=271, S=1505, I=521, N=7910]
================================================================
```

Figura 2.6.3: Resultados modelo de ocho gaussianas

Modelo de 16 gaussianas

```
1235  nano editf8g
1236  mkdir hmm16g.0
1237  mkdir hmm16g.1
1238  mkdir hmm16g.2
1239  HHEd -C ../config/config -T 1 -H hmm8g.2/macros
-H hmm8g.2/hmmdefs -M hmm16g.0 editf16g ../etc/monophones+sil+sp
1240  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm16g.0/macros -H
hmm16g.0/hmmdefs -M hmm16g.1 ../etc/monophones+sil+sp
1241  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm16g.1/macros -H
hmm16g.1/hmmdefs -M hmm16g.2 ../etc/monophones+sil+sp
1246  HVite -C ../config/config -H ../modelos/hmm16g.2/macros
-H ../modelos/hmm16g.2/hmmdefs -S test.scp -l '*' -i
recout16g.mlf -w ../lm/wdnet -p 0.0 -s 5.0 ../etc/dictl40
../etc/monophones+sil+sp &
1249  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout16g.mlf
```

```
---------------------- Overall Results -------------------------
SENT: %Correct=35.36 [H=349, S=638, N=987]
WORD: %Corr=80.57, Acc=75.18 [H=6373, D=263, S=1274, I=426, N=7910]
================================================================
```

Figura 2.6.4: Resultados modelo de dieciseis gaussianas

Modelo de 32 gaussianas

```
1252  mkdir hmm32g.0
1253  mkdir hmm32g.1
1254  mkdir hmm32g.2
1255  nano editf16g
1256  HHEd -C ../config/config -T 1 -H hmm16g.2/macros -H
hmm16g.2/hmmdefs -M hmm32g.0 editf32g ../etc/monophones+sil+sp
1257  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm32g.0/macros -H
hmm32g.0/hmmdefs -M hmm32g.1 ../etc/monophones+sil+sp
1258  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm32g.1/macros -H
hmm32g.1/hmmdefs -M hmm32g.2 ../etc/monophones+sil+sp
1260  HVite -C ../config/config -H ../modelos/hmm32g.2/macros -H
../modelos/hmm32g.2/hmmdefs -S test.scp -l '*' -i recout32g.mlf
-w ../lm/wdnet -p 0.0 -s 5.0 ../etc/dictl40
../etc/monophones+sil+sp &
1261  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout32g.mlf
```

```
---------------------- Overall Results -------------------------
SENT: %Correct=40.73 [H=402, S=585, N=987]
WORD: %Corr=82.81, Acc=78.43 [H=6550, D=230, S=1130, I=346, N=7910]
================================================================
```

Figura 2.6.5: Resultados modelo de treinta y dos gaussianas

Modelo de 64 gaussianas

```
1263  mkdir hmm64g.0
1264  mkdir hmm64g.1
1265  mkdir hmm64g.2
1266  nano editf32g
1267  HHEd -C ../config/config -T 1 -H hmm32g.2/macros -H
hmm32g.2/hmmdefs -M hmm64g.0 editf64g ../etc/monophones+sil+sp
```

```
1268  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm64g.0/macros -H
hmm64g.0/hmmdefs -M hmm64g.1 ../etc/monophones+sil+sp
1269  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm64g.1/macros -H
hmm64g.1/hmmdefs -M hmm64g.2 ../etc/monophones+sil+sp
1272  HVite -C ../config/config -H ../modelos/hmm64g.2/macros -H
../modelos/hmm64g.2/hmmdefs -S test.scp -l '*' -i
recout64g.mlf -w ../lm/wdnet -p 0.0 -s 5.0
../etc/dictl40 ../etc/monophones+sil+sp &
1284  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout64g.mlf
```

```
---------------------- Overall Results ----------------------
SENT: %Correct=44.17 [H=436, S=551, N=987]
WORD: %Corr=84.58, Acc=80.57 [H=6690, D=197, S=1023, I=317, N=7910]
================================================================
```

Figura 2.6.6: Resultados modelo de sesenta y cuatro gaussianas

Modelo de 128 gaussianas

```
1038  mkdir hmm128g.0
1039  mkdir hmm128g.1
1040  mkdir hmm128g.2
1318  nano editf64g
1319  HHEd -C ../config/config -T 1 -H hmm64g.2/macros -H
hmm64g.2/hmmdefs -M hmm128g.0 editf128g ../etc/monophones+sil+sp
1320  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm128g.0/macros -H
hmm128g.0/hmmdefs -M hmm128g.1
../etc/monophones+sil+sp
1321  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm128g.1/macros -H
hmm128g.1/hmmdefs -M hmm128g.2
../etc/monophones+sil+sp
1322  HVite -C ../config/config -H ../modelos/hmm128g.2/macros
-H ../modelos/hmm128g.2/hmmdefs -S test.scp -l '*' -i
recout128g.mlf -w ../lm/wdnet -p 0.0 -s 5.0
../etc/dictl40 ../etc/monophones+sil+sp &
1327  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout128g.mlf
```

```
---------------------- Overall Results -------------------------
SENT: %Correct=46.30 [H=457, S=530, N=987]
WORD: %Corr=85.45, Acc=81.44 [H=6759, D=200, S=951, I=317, N=7910]
================================================================
```

Figura 2.6.7: Resultados modelo de ciento ciento veintiocho gaussianas

Modelo de 256 gaussianas

```
1038  mkdir hmm128g.0
1039  mkdir hmm128g.1
1040  mkdir hmm128g.2
1683  nano editf128g
1690  HHEd -C ../config/config -T 1 -H hmm128g.2/macros -H
hmm128g.2/hmmdefs -M hmm256g.0  editf256g
../etc/monophones+sil+sp
1696  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm256g.0/macros -H
hmm256g.0/hmmdefs -M hmm256g.1 ../etc/monophones+sil+sp
1697  HERest -T 1 -C ../config/config -I ../etc/phones1.mlf -t
250.0 150.0 1000.0 -S train.scp -H hmm256g.1/macros -H
hmm256g.1/hmmdefs -M hmm256g.2 ../etc/monophones+sil+sp
1709  HVite -C ../config/config -H ../modelos/hmm256g.2/macros
-H ../modelos/hmm256g.2/hmmdefs -S test.scp -l '*' -i
recout256g.mlf -w ../lm/wdnet -p 0.0 -s 5.0
../etc/dictl40 ../etc/monophones+sil+sp &
1710  HResults -t -f -I testref.mlf ../etc/monophones+sil+sp
recout256g.mlf
```

```
---------------------- Overall Results -------------------------
SENT: %Correct=43.77 [H=432, S=555, N=987]
WORD: %Corr=85.40, Acc=80.61 [H=6755, D=178, S=977, I=379, N=7910]
================================================================
```

Figura 2.6.8: Resultados modelo de ciento doscientos veintiseis gaussianas

2.7. Resultados

Los resultados del reconocimiento obtenidos aumentando el numero de gaussianas se puede observar de forma resumida a continuación:

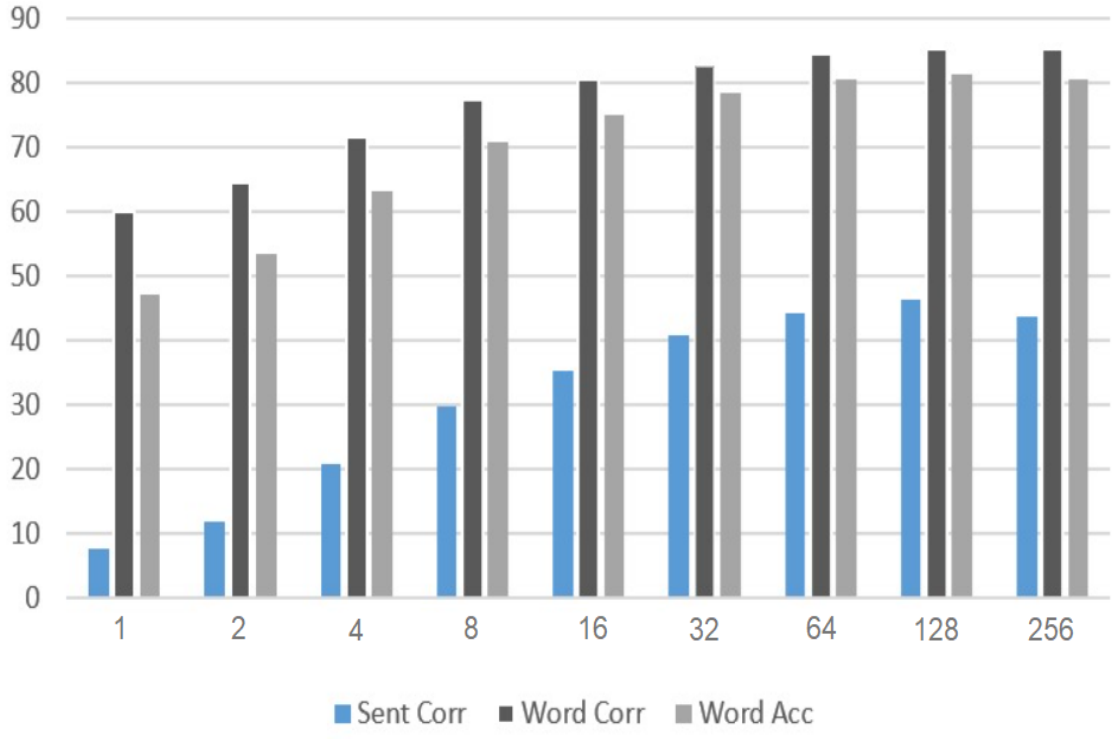

Figura 2.7.1: Resultados reconocimiento

Se nota de forma clara que a medida que se aumenta la cantidad, el resultado mejora hasta llegar un limite sobre 128 gaussianas. A partir de ese punto, si se sigue sumando, el reconocimiento no mejora en ninguna de sus métricas y se tiene un mayor costo computacional, por lo tanto, se concluye que el numero óptimo de gaussianas es de 128.

3. Reconocedor de llamadas telefónicas

En esta sección se hace el desarrollo del reconocedor de gramática finita, en el mismo se utiliza el modelo acústico generado anteriormente para el caso de 128 gaussianas el cual otorgo los mejores resultados.

3.1. Gramática

En primer lugar, para el desarrollo del reconocedor de gramática finita, es necesario definir la gramática a utilizar.

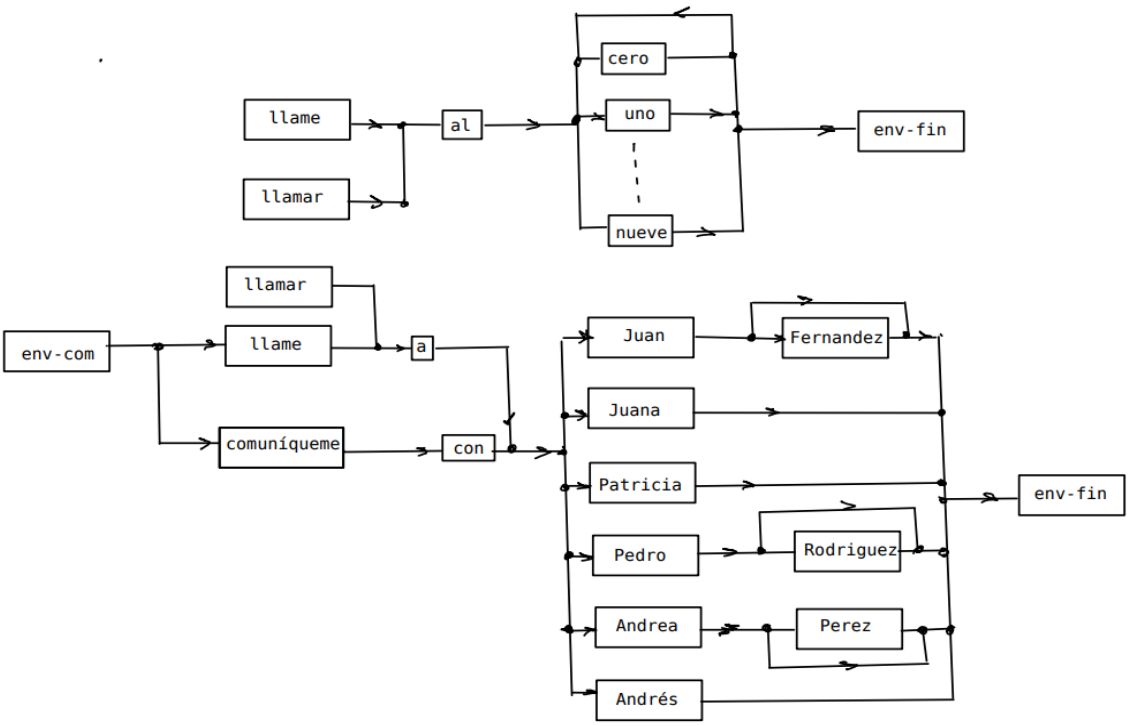

Figura 3.1.1: Gramática finita

Con este objetivo, se crea el archivo *gram* que contiene dos definiciones de variables y una expresión regular.

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

Figura 3.1.2: Contenido archivo gram

En la expresión regular la barra vertical implica que existen alternativas, los corchetes definen elementos opcionales y los signos de mayor y menos indican que puede haber repeticiones.

Posteriormente, con el comando *HParse* se realiza la red de palabras que se almacena en el archivo *wdnet.gf*.

```
1725  nano gram
1731  HParse ../etc/gram wdnet.gf
```

3.2. Diccionario de pronunciaciones

De manera similar a lo que se realizo en el reconocedor de 3000 palabras, se debe crear un diccionario de pronunciaciones de las palabras utilizadas en el reconocedor de gramatica finita.

Se comienza importando el archivo *wlist.gf* que contiene la lista de palabras utilizadas ordenadas alfabéticamente. Seguidamente, se crea el archivo *global.ded.gf* el cual contiene instrucciones que, junto con la herramienta HDMan, generaran el diccionario.

```
AS sp
RS cmu
MP sil sil sp
```

Figura 3.2.1: Contenido archivo global.ded.gf

En la sección **2.3.1** se importo el archivo *lexicon.gf* que contiene cada palabra utilizada con su pronunciación, sin embargo, le falta la pronunciación de la palabra "a", se agrega la misma de forma manual.

Luego, se genera el diccionario de pronunciaciones *dictgf* con la herramienta HDMan. Además, en *hdmangf.log* se guardan los logs donde se muestran estadísticas del proceso y en el archivo *monophones.gf* se genera una lista de todos los fonemas utilizados.

```
1735  cp /home/cestien/Documentos/wlist.gf ./
1740  nano global.ded.gf
1753  nano lexicon.gf
1758  HDMan -m -w wlist.gf -n monophones.gf -l
../log/hdmangf.log dictgf lexicon.gf
```

Por ultimo, se agrega el fonema /sil/ (silencio largo) al archivo monophones.gf y los fonemas de comienzo y fin de frase los cuales tienen asociados un silencio largo.

```
1781  echo "sil" >> monophones.gf
1787  echo "env-com [] sil" >> dictgf
1788  echo "env-fin [] sil" >> dictgf
```

3.3. Generación de frases

Una vez que ya se tiene el diccionario construido y la lista de palabras a utilizar, se generan con la herramienta *HSGen* doscientas frases aleatorias que son almacenadas en el archivo *promptsgf.test*.

```
1793  HSGen -l -n 200 ../lm/wdnet.gf dictgf > promptsgf.test
```

Posteriormente, se procede a grabar los audios de cada una de las 200 frases en formato wav mono de 16khz de frecuencia. Se crean los directorios necesarios para almacenarlos.

```
1799  mkdir wavgf
1800  mkdir mfcgf
```

Finalmente, se utiliza la herramienta HCopy para generar los archivos mfc correspondientes a las grabaciones en formato wav. Como se explico previamente esta herramienta requiere de dos archivos; El primero, *configgf.hcopy*, contiene la misma configuración que en el caso anterior pero se le modifica el *Source format* a tipo *Wave*. El segundo,*genmfcgf.test*, corresponde con los pares de nombres de archivos wav-mfc.

```
1850  cp config.hcopy configgf.hcopy
1853  nano configgf.hcopy
1870  nano genmfcgf.test
1962  HCopy -T 1 -C ../config/configgf.hcopy -S genmfcgf.test
```

3.4. Reconocimiento

En este inciso se realizara el reconocimiento de las frases grabadas. Para el mismo, se utiliza el algoritmo de Viterbi aplicado con el modelo de mezcla de 128 gaussianas.

Se comienza guardando las rutas a los archivos mfc en el archivo *testrefgf.scp* generado con la siguiente instrucción:

```
1972  ls ../datos/mfcgf/* > testrefgf.scp
```

Luego se aplica el algoritmo de viterbi con la herramienta HVite, al igual que en el reconocedor de 3000 palabras. El mismo utiliza el archivo que contiene las rutas a los archivos mfc recién generado, la red de palabras para gramática finita, la lista de monofonos y el diccionario de pronunciaciones generado en la sección **3.2**.

```
1990  HVite -C ../config/config -H ../modelos/hmm128g.2/macros
-H ../modelos/hmm128g.2/hmmdefs -S testgf.scp -l '*' -i
recoutgf.mlf -w ../lm/wdnet.gf -p 0.0 -s 5.0 ../etc/dictgf
../etc/monophones.gf &
```

Por ultimo, se genera un archivo *testrefgf.mlf* que contiene la traducción a nivel texto del contenido de las grabaciones y se aplica la herramienta de análisis de resultados HResult:

```
1991  HResults -t -f -I testrefgf.mlf ../etc/monophones.gf
recoutgf.mlf
```

La herramienta tiene la capacidad de comparar dos juegos de Label Files (recoutgf.mlf y testrefgf.mlf) y devolver las métricas de resultado que se explicaron anteriormente. Se llega al resultado que se muestras a continuación:

```
---------------------- Overall Results ----------------------
SENT: %Correct=75.50 [H=151, S=49, N=200]
WORD: %Corr=96.78, Acc=90.35 [H=1505, D=3, S=47, I=100, N=1555]
================================================================
```

Figura 3.4.1: Resultados reconocedor de gramática finita

3.5. Conclusión

Tras el análisis de los resultados, se puede concluir que la performance del reconocedor de habla continua es correcta, se obtuvo un acierto de palabras del 96,78 % superando así el 90 % requerido.

En adición, comparando estos resultados con el del reconocedor de 3000 palabras, se puede observar que dan un porcentaje de acierto mucho mayor. Esto se debe a que la cantidad de palabras del diccionario es menor, y las mismas, tienen restricciones (gramática finita) haciendo que las posibilidades de elección de la palabra sean menos y así, el reconocedor otorgue mejores resultados.

Por ultimo, se concluye que para el caso de este reconocedor el modelo que entrega mejores resultados es el generado con una mezcla de 128 distribuciones gaussianas.