UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

RECONOCIMIENTO CONTINUO DEL ESPAÑOL HABLADO EN MÉXICO

T E S I S

QUE PARA OBTENER EL TÍTULO DE

MAESTRO EN INGENIERÍA

AREA: ELÉCTRICA.

CAMPO: PROCESAMIENTO DIGITAL DE SEÑALES.

P R E S E N T A:

JAIME ALFONSO REYES CORTÉS

TUTOR: DR. JOSÉ ABEL HERRERA CAMACHO

CIUDAD UNIVERSITARIA MÉXICO, D.F. 2010

JURADO ASIGNADO:

Presidente:	DR. BOHUMIL PSENICKA
Secretario:	DR. JESÚS SAVAGE CARMONA
Vocal:	DR. JOSÉ ABEL HERRERA CAMACHO
1er. Suplente:	DR. GERARDO EUGENIO SIERRA MARTÍNEZ
2do. Suplente:	DR. ALFONSO MEDINA URREA

Lugar o lugares donde se realizó la tesis:

MÉXICO, D.F.FACULTAD DE INGENIERIA, UNAM. SECRETARIA DE POSGRADO E INVESTIGACIÓN. LABORATORIO DE PROCESAMIENTO DE VOZ.

TUTOR DE TESIS:

DR .JOSÉ ABEL HERRERA CAMACHO

FIRMA

Dedicatoria

A mi abuelita, **Josefina Cedillo Granados** (QEPD), porque me mostraste mis raíces y toda la fortaleza que nos heredaste. Aún con tu dolorosa pérdida nos hiciste ver que podemos continuar a pesar de todo. Estarás en mi memoria y en mi corazón por siempre.

A mi madre, **María Modesta Cortés Cedillo**, que gracias a tu férrea voluntad y a tu carácter inquebrantable me has hecho ver que toda situación "es difícil, pero no imposible".

Al amor de mi vida, **Inés Jiménez Chavarria**, porque que me has apoyado siempre, porque me abriste tu corazón y con tu noble carácter y tu gentileza haces que el mundo sea un lugar mejor.

Agradecimientos

A mi padre, Crescencio Reyes Soto, por tu gran apoyo y aún cuando el cansancio te invadió en algún momento, me acompañaste en este trayecto sin vacilar.

A mis hermanos, Héctor Omar y Aldo Reyes Cortés, por su gran apoyo y porque siempre hacen que me ría, hasta de mí mismo.

A mi *alma mater*, la UNAM, que, gracias a su naturaleza, me ha otorgado una visión universal de las cosas y me ha dado mucho más que una formación académica.

A la Facultad de Ingeniería por brindarme todas sus instalaciones para tener un mejor desempeño en mi profesión y superarme día a día.

A mi tutor, el Dr. Abel Herrera Camacho, por su paciencia, su apoyo y todos sus consejos para poder llevar a cabo este trabajo.

A mis profesores por brindarme sus conocimientos y sus consejos incondicionalmente y por motivarme a seguir en esta área.

A la Dra. Ana María Vázquez Vargas por creer en mís aptitudes para realizar este trabajo.

Al CEP por la beca otorgada durante 3 semestres para realizar mis estudios de Maestría.

Al Grupo de Ingeniería Lingüística por la beca otorgada durante 6 meses, como parte del proyecto DGAPA PAPIIT IN402008: Glutinometría y Variación Dialectal, para concluir con mi tesis de Maestría.

A mi prima, Cintia Quezada Reyes, por tu valiosa amistad y porque sembraste en mí la semilla de la investigación.

A las ingenieras Laura, Jaqui y Maricela, que me han brindado su amistad y sus valiosos consejos.

A Omar Nieto Crisóstomo por tu amistad, tu compañerismo, por tus valiosos consejos y porque me motivaste a seguir en la investigación.

A Oscar Navarrete Tolento por toda tu amistad y apoyo durante nuestra estancia en el Laboratorio de Procesamiento de Voz.

A Josefina y Nato, por su valiosa amistad y su invaluable apoyo en el Laboratorio de Multimedia.

A la Danza, porque al encontrarla cambió mi vida y me dió una perspectiva diferente del mundo.

A mis amigos, que, aún en la distancia, me han brindado su apoyo y cariño.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Introducción

El procesamiento automático de voz

Para la mayoría de la gente, la voz, es la forma más natural de intercambiar información. La meta de la tecnología de reconocimiento de voz, en sentido amplio, es crear máquinas que puedan recibir información hablada y que sean capaces de actuar apropiadamente con base en dicha información. Además, la información extra que pueda requerir la computadora de parte del usuario se solicitaría utilizando síntesis de voz. Desde este punto de vista, el reconocimiento de voz es parte de la inteligencia artificial, es decir, máquinas que sean capaces de escuchar, entender y actuar con la información hablada que se les proporcione y también que sean capaces de hablar para completar el intercambio de información [9]. Cualquier persona que haya leído o visto historias de ciencia ficción sabe que la capacidad humana para imaginar la forma y las aplicaciones para esta tecnología no tiene límites. Sin embargo, nuestra imaginación a veces sobrepasa nuestras habilidades técnicas en este dominio para el cuál, ya en pleno siglo XXI, se encuentra todavía limitado, puesto que el objetivo de que exista una máquina parlante que sea robusta, inteligente y que converse fluidamente acerca de cualquier tema aún permanece un tanto distante. Actualmente lo que se comprende y se sabe acerca de la tecnología de voz se está incrementando a un ritmo considerable, sin embargo, el conocimiento actual, a pesar de que es aplicado a tareas ya no tan modestas, como la realización de un dictado, aún se sigue aplicando a dominios restringidos.

Hay que hacer la distinción entre dos procesos: Por *reconocimiento automático de voz* (*Automatic Speech Recognition, ASR*) se entiende al proceso por el cual una computadora establece una correspondencia entre una señal de voz acústica con su texto correspondiente y por *comprensión automática de voz* (*Automatic Speech Understanding, ASU*) se entiende al proceso por el cual una computadora establece una correspondencia entre una señal de voz acústica con alguna forma de significado abstracto de la voz [17]. Así tenemos varios tipos de sistemas de reconocimiento automático de voz. Por una parte se tiene a los *sistemas de reconocimiento de voz dependientes del locutor*, que están orientados para trabajar con un solo hablante. Estos sistemas generalmente son más fáciles de desarrollar, más baratos y más precisos, sin embargo no

son tan flexibles como los sistemas de reconocimientos adaptables al locutor e independientes del locutor. También existen los *sistemas de reconocimiento de voz independientes del locutor*, que están orientado para operar con cualquier hablante de un determinado tipo, por ejemplo un hablante del español de la ciudad de México. Estos sistemas son los más difíciles de desarrollar, ya que son más costosos y su nivel de precisión es más bajo que un sistema dependiente de locutor, sin embargo, poseen mayor flexibilidad que los anteriores. Existen también los *sistemas de reconocimiento de voz adaptables al locutor* que están orientados para ajustar su operación a las características de nuevos locutores. En cuanto a nivel de dificultad de desarrollo, se encuentran en un nivel intermedio entre los sistemas independientes del locutor y los dependientes del locutor.

En cuanto a complejidad se refiere, existen los *sistemas de reconocimiento de palabras aisladas*, que operan con una palabra a la vez, es decir, requieren que se haga una pausa entre cada palabra pronunciada. Este tipo de sistemas lleva a cabo la forma más simple de reconocimiento de voz porque los límites de las palabras son más fáciles de encontrar y la pronunciación de una palabra no afecta a las otras. Además, la ocurrencia de las palabras es más consistente, lo que hace que las palabras sean más fáciles de reconocer.

Los sistemas de reconocimiento más complejos son los *sistemas de reconocimiento de voz continua* (*Continuous Speech Recognition, CSR*) en los cuáles el usuario pronuncia un mensaje de manera relativa o completamente sin restricciones. En primer lugar, el reconocedor, de alguna manera, debe ser capaz de tratar con los límites temporales de una señal acústica que son desconocidos. En segundo lugar, el reconocedor debe ser capaz de tener un buen desempeño en la presencia de todos los efectos coarticulatorios y las variaciones de articulación, incluyendo inserciones y omisiones de palabras, que acompañan al lenguaje natural. A diferencia de los sistemas de reconocimiento de palabras aisladas, los sistemas de reconocimiento de voz continua no deben requerir de la cooperación del locutor; deben compensar este hecho al emplear algoritmos más robustos que puedan manejar la gran cantidad de pequeñas variaciones en el lenguaje fluido. Los sistemas CSR representan una manera más natural de hablar desde el punto de vista del usuario y son esenciales en muchas de las aplicaciones en las que gran cantidad de gente interactúa con el reconocedor. Los sistemas CSR con vocabulario extenso deben ser entrenados con unidades en subpalabras, como fonemas, semisílabas o sílabas, y las relaciones entre palabras deben explotarse para lograr un buen desempeño. Existen mayores requerimientos en el caso de voz continua, ya que al modelar las palabras se deben capturar las variaciones fonológicas dentro de las palabras y entre las mismas y, quizás, aprender y explotar las relaciones probabilísticas entre unidades de subpalabras (conocimientos léxico y fonológico), así como las relaciones entre palabras (sintaxis), su significado (semántica) y su pronunciación (prosodia) a una escala de análisis mucho mayor.

Arquitectura de un sistema de reconocimiento de voz

La mayoría de los sistemas de reconocimiento de voz actuales presentan una arquitectura como la que se representa en la figura 1 y que está delimitada por la línea punteada.
Las aplicaciones se comunican con el decodificador para obtener los resultados del reconocimiento y que pueden ser utilizados para adaptar otros componentes en el sistema. Los modelos acústicos pueden incluir la representación

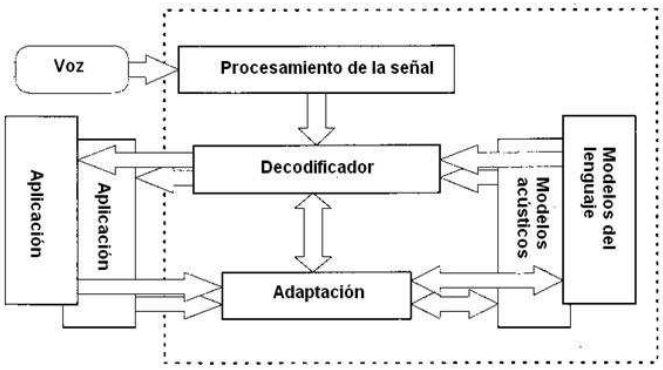


Figura 1: Arquitectura funcional de un sistema de Reconocimiento de voz [15]

del conocimiento como acústica, fonética, la variabilidad del ambiente y del micrófono, el género, las diferencias de acento entre los parlantes, entre otros. Los modelos del lenguaje se refieren al *conocimiento del sistema* de lo que constituye una posible palabra, qué palabras son parecidas y pueden ocurrir en la frase y en qué secuencia. La semántica y las funciones relacionadas con una operación que el usuario puede desear llevar a cabo también pueden ser necesarias para el modelo del lenguaje. Existe bastante incertidumbre en estás áreas que se encuentran asociadas con las características del parlante: el estilo de la voz y la velocidad, el reconocimiento de segmentos básicos de voz, las posibles palabras, las palabras parecidas, las desconocidas, sus variaciones gramaticales, la interferencia del ruido, los acentos dependiendo de la región y la precisión de los resultados. Un sistema de reconocimiento confiable y eficiente debe tratar con todas estas incertidumbres. Pero esto es sólo el principio. La incertidumbre acústica de los diferentes acentos y las formas de hablar de cada parlante están compuestos por complejidades léxicas y gramaticales y por las variaciones del lenguaje hablado, todas ellas deben ser representadas en el modelo del lenguaje.

Problemas que presenta un sistema de reconocimiento de voz

Las señales de voz tienden a concatenarse, es decir, cuando nos comunicamos es difícil encontrar dónde se inicia una palabra y dónde termina

dicha palabra y es difícil delimitar los sonidos que componen dicha palabra, aunque la experiencia ha demostrado que basta con una separación aproximada.

Las señales de voz son variables, es decir, dos señales de voz de la misma frase e incluso de la misma palabra resultan ser distintas aún cuando hayan sido pronunciadas por el mismo locutor.

Existe ambigüedad en el habla, esto es, existen palabras cuya pronunciación es igual, pero su significado es diferente (homófonas), además, una misma palabra puede tener varios significados dependiendo del contexto en el que se hable.

Presencia de ruido en las señales de voz, debido a que en el ambiente en que se trabaja existen ciertas perturbaciones que tienden a modificar las señales, haciéndolas difíciles de reconocer.

Complejidad en el habla, ésta varía según el acuerdo entre los individuos, la región en la que se habite, con quién se esté hablando e incluso el estado de ánimo en el que se encuentre el parlante, además de que solo es una parte del proceso de comunicación.

El tamaño del vocabulario o corpus de un sistema de reconocimiento de voz afecta su complejidad, sus requerimientos de procesamiento y la precisión del sistema. Vagamente hablando los sistemas de reconocimiento de voz actuales en cuanto a la cantidad de palabras que manejan en su vocabulario y que tienen un desempeño aceptable se centran en alguna de las siguientes categorías:

- Aquellas con pequeños vocabularios (10 a 100 palabras)
- Aquellos en las cuáles las palabras son habladas en aislamiento deliberadamente (vocabularios que pueden exceder las 10000 palabras)
- Aquellas que aceptan voz continua pero que están acotados a tareas de dominios específicos, como por ejemplo mensajes que ocurren en una oficina de correspondencia en particular (típicamente vocabularios de 1000 a 5000 palabras)
- Sistemas de reconocimiento de oraciones basadas en trigramas con ausencia de ruido (su vocabulario es de 20000 palabras o más).

Un sistema de reconocimiento de voz tendrá éxito en la medida que resuelva los problemas planteados anteriormente [39].

El sistema *Sphinx*

Sphinx es uno de los sistemas de reconocimiento de voz más versátiles que existen en el mundo. Fue desarrollado por el grupo *Sphinx* en la Universidad de Carnegie Mellon [48] y se basa en la construcción de Modelos Ocultos de Markov.

Sphinx es un sistema de reconocimiento de voz continua, de gran vocabulario e independiente del locutor que está realizado bajo la licencia BSD [1]. Posee una colección de herramientas y recursos en código abierto, conocidos como el proyecto *CMU Sphinxproject*, que permiten a los investigadores y desarrolladores construir sistemas de reconocimiento de voz.
El proyecto *CMU Sphinxproject* desarrolla una serie de reconocedores de voz para construir aplicaciones de reconocimiento de voz de alto desempeño. En los últimos años también incluye recursos relacionados con el reconocimiento de voz, tales como un entrenador acústico, un entrenador del modelo del lenguaje así como modelos acústicos pre-entrenados [6].
Actualmente existen varios proyectos desarrollados en CMU Sphinx:

1. **Sphinx 2**. Un reconocedor de voz de gran vocabulario y de alta velocidad. Generalmente utilizado en sistemas de diálogo y sistemas de aprendizaje de pronunciación

2. **Sphinx 3**. Un reconocedor de voz ligeramente más lento que *Sphinx 2* pero más preciso. Generalmente se utiliza como una implementación de servidor de Sphinx o para evaluación.

3. **Sphinx 4**. Una versión completa de Sphinx escrita en Java. Proporciona alta precisión y la velocidad de su desempeño es comparable al estado del arte.

4. **PocketSphinx**. Un reconocedor de voz que puede ser utilizado en sistemas embebidos.

5. **SphinxTrain**. Una suite de herramientas que llevan a cabo entrenamiento del modelo acústico. Contiene recetas para el entrenamiento.

6. **CMU-Cambridge Language Modeling Toolkit, CMU-CLM**. Una suite de herramientas que llevan a cabo entrenamiento del modelo del lenguaje.

7. **SphinxBase**. Un conjunto de librerías comúnes utilizadas para varios proyectos de CMU Sphinx.

Para la realización del presente trabajo me apoyé de *Sphinx 3* y de las herramientas de *CMU-CLM* debido, por una parte, a que en el laboratorio de Procesamiento de Voz del Posgrado de Ingeniería Eléctrica de la UNAM se cuenta con una máquina prestada por la Universidad de Carnegie Mellon y que tiene instalada esta versión, también a que mi meta es la de diseñar un sistema de reconocimiento de voz continua del español para no depender de ningún software extranjero, y por otra, *Sphinx* es un software de código abierto lo que me dió la posibilidad de analizarlo para desarrollar mis propios módulos.

Cabe mencionar que los términos *Sphinx 3* y *Sphinx* se utilizarán indistintamente haciendo alusión a la versión *Sphinx 3*.

Los componentes principales del sistema *Sphinx* son el módulo *Sphinxtrain*, utilizado para entrenamiento de los modelos acústicos, y el módulo *Sphinx decoder*, utilizado para el reconocimiento de la voz.

El módulo *Sphinxtrain* se encarga de la obtención de los parámetros de los modelos de las unidades de sonido que se obtienen por medio de muestras de la señal de voz; a estas muestras se les denomina *bases de entrenamiento*. El módulo de entrenamiento también requiere que se le indique qué unidades de sonido se desean entrenar y al menos la secuencia en la cual ocurre cada señal de voz dentro de la base de entrenamiento. Esta información se le proporciona al módulo Sphinxtrain a través de un *archivo de transcripción*, en el que se indican tanto la secuencia de palabras como los sonidos que no forman parte del discurso, como disfluencias, pausas, etcétera, en el orden exacto como ocurren dentro de la señal de voz, seguidos por una etiqueta que puede ser utilizada para asociar esta secuencia con su correspondiente señal de voz. Enseguida, el módulo de entrenamiento busca en un *diccionario de lenguaje*, que relaciona cada palabra con una secuencia de unidades de sonido, con el fin de obtener la secuencia de unidades de sonido que están asociadas con cada señal de voz. Asimismo se ayuda de un *diccionario de relleno* para establecer la correspondencia entre los sonidos que no forman parte del discurso.

Por su parte, el módulo *Sphinx decoder*, dadas las entradas adecuadas, realiza la tarea de reconocimiento. Las entradas necesarias son: los modelos acústicos entrenados, un archivo de *índices del modelo*, el modelo del lenguaje, un diccionario del lenguaje, un diccionario de relleno y un conjunto de señales acústicas que se desea reconocer. A los datos que se desea reconocer se les denomina *conjunto de prueba*. Con los modelos acústicos entrenados, el módulo de entrenamiento generará archivos de índices del modelo. Un archivo de índices del modelo simplemente contiene identificadores para cada estado de cada modelo oculto de Markov, los cuales son usados por los módulos de entrenamiento y de reconocimiento para tener acceso al conjunto correcto de parámetros para aquellos estados de los modelos ocultos de Markov. Con cualquier conjunto de modelos acústicos, se debe utilizar su archivo de índices de modelo para realizar la decodificación.

El presente trabajo cubre los aspectos necesarios para llevar a cabo el reconocimiento de voz continua del español hablado en la región central de México. Para ello me apoyo del sistema *Sphinx*, ajustando sus módulos para realizar las tareas de entrenamiento y de reconocimiento.
En primer lugar se partirá con los Modelos Ocultos de Markov (Capítulo 1), que es una de las técnicas de clasificación de patrones que da sustento al funcionamiento de muchos de los sistemas de reconocimiento de tipo estocástico. En los capítulos 2 y 3 se darán las bases de los modelados acústico y del lenguaje, respectivamente.
En el capítulo 4 se describirá con más detalle el proceso de decodificación que se emplea en el sistema *Sphinx* y en el capitulo 5 se explicarán los módulos y

procedimientos empleados en el sistema *Sphinx* para que, finalmente se observen los resultados del entrenamiento y del reconocimiento así como las conclusiones plasmados en el presente trabajo (Capítulos 6 y 7).

1

Clasificación de patrones

El reconocimiento de patrones es la técnica más específica de todo sistema de reconocimiento. De ahí que muchos reconocedores se identifiquen a partir de la técnica de reconocimiento de patrones que incorporan. A partir de la representación paramétrica de la voz, el módulo de clasificación de patrones realiza un proceso de clasificación utilizando una serie de patrones. Dichos patrones se obtienen en una fase de entrenamiento del sistema y son representativos de un conjunto de unidades lingüísticas, como palabras, sílabas, sonidos o fonemas. La peculiaridad más característica de este proceso, que marca su dificultad, es la variabilidad temporal que puede presentar una misma unidad lingüística al ser producida por medio de diferentes modos y/o de diferentes velocidades del habla. Así, las primeras técnicas de reconocimiento de patrones utilizadas fueron las basadas en un Alineamiento Temporal a través de algoritmos de Programación Dinámica y técnicas de Ajuste Dinámico en el Tiempo (Dynamic Time Warping, DTW). Posteriormente se recurrió a otra alternativa, basada en el modelado de procesos estocásticos, que permite representar secuencias de duración variable con mayor flexibilidad. Concretamente, la alternativa a las técnicas DTW fueron los Modelos Ocultos de Markov (Hidden Markov Models, HMM) que pueden verse como una generalización del algoritmo DTW y que han demostrado tener mejores prestaciones en multitud de sistemas de reconocimiento. Otra posible alternativa frente a los HMM son las Redes Neuronales Artificiales (Artificial Neural Networks, ANN), sin embargo, en este trabajo no se tratarán las técnicas DTW ni ANN.

1.1. Componentes de un sistema de reconocimiento automático de voz

Un sistema canónico que utiliza la aproximación del reconocimiento de patrones en ASR se describe en la figura 1.1.

Tal sistema opera en dos fases [19]:

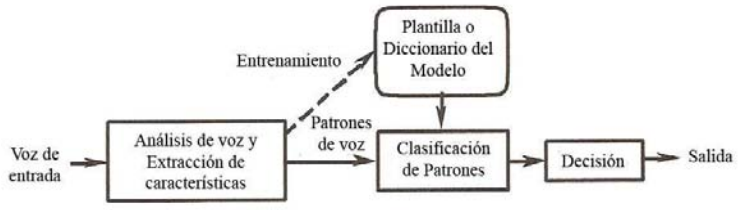


Figura 1.1: Esquema de un reconocedor de voz que se basa en el principio de reconocimiento de patrones [19].

1. *Fase de entrenamiento*, durante la cual el sistema "aprende" o es "alimentado" con los patrones de referencia que representan a las distintas unidades de voz, frases, palabras o fonos, que constituyen el vocabulario de la aplicación. Cada referencia se toma a partir de ejemplos hablados y se almacena ya sea a manera de plantillas, que se obtienen a partir de algún método que obtiene la plantilla promedio, o de modelos que caracterizan las propiedades estadísticas de los patrones, tales como los sistemas estocásticos. En ambos casos, la fase de aprendizaje necesita algoritmos de aprendizaje eficientes que le proporcionen al sistema los patrones de referencia que sean realmente representativos.

2. *Fase de reconocimiento* durante la cual un patrón desconocido de entrada es identificado al considerar el conjunto de patrones de referencia. El proceso de reconocimiento consiste, a su vez, de tres pasos:

   - *extracción de características*: este primer paso se enfoca en la obtención de un conjunto de parámetros obtenidos por algún método de análisis de señales a partir de porciones sucesivas de la señal de voz. En el presente trabajo la extracción de características se lleva a cabo por medio de coeficientes Mel Frequency Cepstral o MFCC.
   - *clasificación de patrones*: durante este paso se calcula una medida de similitud o *distancia* entre la voz de entrada y cada patrón de referencia. Este proceso requiere de definir una medida local de distancia entre vectores de características y un método para alinear los dos patrones de voz, que puede diferir de acuerdo a la duración y a la tasa del habla (speech rate).
   - *decisión*: durante este paso final, al patrón desconocido se le asigna una etiqueta que se relaciona con el patrón de referencia más cercano. Esta decisión está basada en algunas reglas que toman en cuenta los resultados de las medidas de similitud.

Los sistemas de reconocimiento de voz poseen algunas características en común: En primer lugar, las técnicas son independientes del tipo de clases de voz utilizadas, por lo que es posible desarrollar sistemas capaces de reconocer palabras,

frases o subpalabras, tales como sílabas o fonemas. En segundo lugar, puesto que en estos reconocedores genéricos no es necesario de conocimientos de voz explícitos, un sistema puede ser usado con diferentes vocabularios, tareas e incluso con diferentes lenguajes, como el español, el inglés o el francés. Por último, pero sin restarle importancia, la precisión de tales sistemas es altamente dependiente de la cantidad de datos de entrenamiento: mientras mayor sea la cantidad de datos que se tengan, la precisión del reconocedor será mucho mejor. Además la calidad de los datos de entrenamiento también es de gran importancia; en particular, la relación pobre o inapropiada que se establezca entre las condiciones de entrenamiento y las condiciones de prueba es un factor principal en la degradación del desempeño de tales sistemas.

1.2. Modelos de Markov

Considérense aproximaciones estadísticas puras basadas en procesos estocásticos; específicamente hablando, los procesos de Markov. Estas aproximaciones estadísticas pertenecen a un largo y extenso conjunto de *teorías de decisión estadística*, que proporcionan un marco tanto para modelar patrones en forma estadística como para formalizar el proceso de toma de decisiones en forma tal que la pérdida promedio por decisión sea lo más pequeña posible. Las redes son la técnica más utilizada en muchos sistemas ASR para representar información acerca de eventos acústicos en la voz. Por ejemplo, el conocimiento acerca de restricciones de sintaxis y de semántica en secuencias de palabras que son permitidas puede ser codificada en forma eficiente mediante una red cuyos estados son las palabras del vocabulario. Las transiciones entre estados se permiten sólo si la cadena de palabras resultante genera una oración válida siguiendo la gramática del sistema.
Una *cadena de Markov* consiste de un conjunto de estados, con transiciones entre estados. A cada estado le corresponde un símbolo y a cada transición se le asocia una probabilidad, como se ve en la figura 1.2. Los símbolos se consideran como la salida del modelo de Markov y se producen como resultado de las transiciones de probabilidad al pasar de un estado a otro. Tal comportamiento conduce a que estos modelos sean utilizados para estudiar fenómenos en los que símbolos de caracter determinista son observados y ordenados en series de tiempo. Generalmente se trabajan con *procesos de Markov de primer orden* o *cadenas de Markov de primer orden*. Una cadena de Markov de primer orden es aquel proceso donde la probabilidad de estar en un estado dado únicamente depende del estado inmediato anterior
Una cadena de Markov es un modelo bastante restrictivo para estudiar problemas más complejos como aquellos asociados con el reconocimiento de voz. Por tal motivo, es necesario extender el modelo para que sea capaz de tratar los casos en que las observaciones sean funciones probabilísticas de los estados. Esto conduce a una formulación dual: las observaciones de voz pueden ser generadas a partir de los estados o de las transiciones.

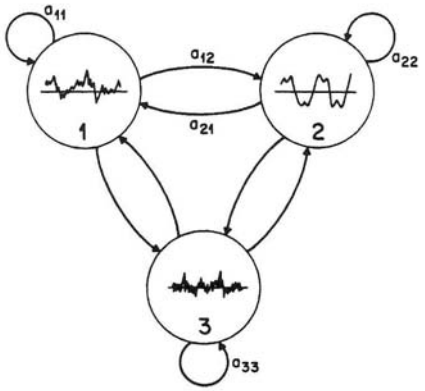


Figura 1.2: Cadena de Markov [11].

1.2.1. Modelos ocultos de Markov (HMM)

Un modelo oculto de Markov es similar a una cadena de Markov, excepto porque los símbolos de salida son probabilísticos. De hecho, la ocurrencia de todos los símbolos es posible en cada estado y cada símbolo tiene asociada su propia probabilidad, y por tanto, a cada estado se le asocia una distribución de probabilidad de todas los símbolos posibles. En otras palabras, un HMM se compone de un *proceso no observable* u *oculto*, una cadena de Markov, y un *proceso de observación* que liga a los vectores acústicos, extraídos de una señal de voz, con los estados del proceso oculto. Por ello, a un HMM también se le conoce como un *proceso doblemente estocástico*. En la figura 1.3 se muestra un

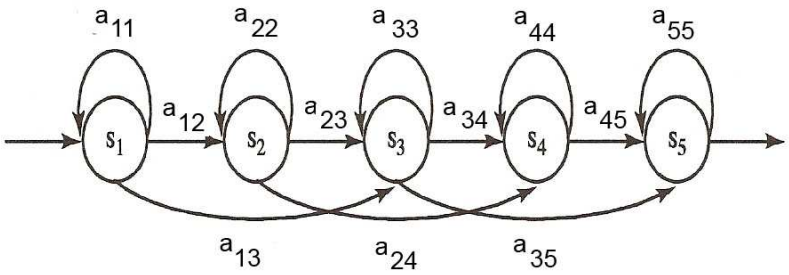


Figura 1.3: HMM de Bakis [9].

HMM de cinco estados que representa una unidad de voz (fonema, palabra, etc.) con transiciones permitidas. Este grafo puede ser visto como un modelo de producción en el que cada transición corresponde a la emisión de una trama de voz o vector de características. A cada estado s_j le corresponde una distribución de probabilidad $P(e_k|s_j)$, que representa la probabilidad de producir el evento e_k cuando una transición desde este estado ocurra; y a cada arco le corresponde

una probabilidad $P(s_j|s_i) = a_{ij}$, que representa la probabilidad de transición del estado s_i al estado s_j. Debido a que en la voz existen restricciones temporales, se utiliza un HMM de tipo Bakis o modelo oculto de Markov de izquierda a derecha.
Un HMM puede representar una unidad de voz, como una subpalabra, una palabra e incluso una oración completa. En sistemas de reconocimiento de gran vocabulario, los HMM generalmente representan unidades de subpalabras, como los fonemas, ésto con el fin de limitar la cantidad de datos de entrenamiento y el espacio de almacenamiento que se requiere para modelar a las palabras. Contrariamente, en sistemas de pequeño vocabulario, la tendencia es utilizar HMM para representar palabras completas.
Para definir un HMM considérese la siguiente notación:

- $\mathbf{O} = O_1, O_2, \ldots, O_T$ es una secuencia de observaciones de la señal de voz;
- $\mathbf{N}$ indica el número de estados;
- $\mathbf{\Omega} = s_1, s_2, \ldots, s_N$ es un conjunto de estados del modelo;
- $\mathbf{A} = a_{ij}$ es la matriz de probabilidades de transición del estado i al estado j;
- $\mathbf{B} = b_i(O_t)$ es la matriz de probabilidades del estado i emitiendo el vector de salida O_t;
- $\pi = \pi_i$ son las distribuciones de probabilidad de los estados iniciales;
- λ representa el conjunto de parámetros que definen al HMM: las matrices de probabilidades de transición, A, las probabilidades de emisión, B, y las distribuciones de probabilidad de los estados iniciales, π, es decir, $\lambda = (\mathbf{A}, \mathbf{B}, \pi)$.

Para cualquier clase de HMM existen tres problemas fundamentales que deben ser resueltos para que los modelos sean puestos en práctica:

1. *El problema de evaluación.* Dado un HMM λ y una secuencia de observaciones de la señal de voz, $O = O_1, O_2, \ldots, O_T$, ¿cuál es la probabilidad de la secuencia de observación, condicionada en el modelo? Una solución eficiente a este problema se da con la etapa *hacia adelante* (*forward*) del *algoritmo hacia adelante-hacia atrás*(*forward-backward*).

2. *El problema de decodificación.* Dado un HMM λ y una secuencia de observaciones, $O = O_1, O_2, \ldots, O_T$, ¿cuál es la secuencia de estados en el modelo que mejor explique tales observaciones? La solución a este problema requiere de un criterio de optimización para encontrar la mejor solución que sea posible. Típicamente, se hace uso del algoritmo de Viterbi.

3. *El problema de aprendizaje.* Dado un HMM λ y una secuencia de observaciones, $O = O_1, O_2, \ldots, O_T$, ¿de qué manera ajustamos los parámetros del modelo de forma tal, que se maximize la probabilidad de generar las

observaciones (criterio de Máxima Similitud o ML)? La secuencia de observación es llamada la secuencia de entrenamiento. La fase de entrenamiento es crucial en el diseño de un sistema automático de reconocimiento de voz basado en HMM, ya que hace posible que los parámetros del modelo sean adaptados en forma óptima a un fenómeno del mundo real.

Las formulaciones matemáticas que se presentan continuación se consideran para el caso en que la secuencia de observaciones sea un conjunto discreto de símbolos. Sin embargo, la solución de los tres problemas de los HMM puede ser extendida al caso en el que las observaciones sean vectores continuos multidimensionales, como se tratará en la sección 1.2.2.

El problema de evaluación

El algoritmo hacia adelante-hacia atrás es un procedimiento eficiente para calcular $P(O|\lambda)$, la probabilidad de la secuencia de observación O, dado el modelo λ.
Sea la *variable hacia adelante* (*forward variable*) $\alpha_t(i)$, definida como

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda) \tag{1.1}$$

es decir, la probabilidad conjunta de observar los primeros t vectores acústicos, estando en el estado S_i durante el instante t, dado el modelo λ. $\alpha_t(i)$ puede ser calculada por medio del procedimiento hacia adelante (forward) dado en el algoritmo 1.1. En el paso de inicialización se establecen las probabilidades hacia delante como el producto de la probabilidad de estar en el estado inicial i y la probabilidad del estado i emitiendo el vector de salida O_1. El paso de *inducción*, es el corazón del cálculo; la figura 1.4 muestra como el estado S_j puede ser alcanzado en el instante $t+1$, a través de los N posibles estados S_i, donde $1 \leq i \leq N$, en el instante t. Debido a que $\alpha_t(i)$ es la probabilidad del evento conjunto donde $O_1, O_2, \cdots, O_t$ son observados, y el estado en el instante t es S_i; el producto $\alpha_t(i)a_{ij}$ es entonces, la probabilidad de que el evento conjunto $O_1O_2\cdots O_t$ sea observado, y el estado S_j sea alcanzado en el instante $t+1$ por conducto del estado S_i en el instante t. Sumando este producto sobre todos los N posibles estados S_i, $1 \leq i \leq N$ en el instante t, resulta en la probabilidad de S_j en el instante $t+1$ con todo el acompañamiento previo de las observaciones parciales [35]. Una vez que ésto se ha obtendido y que S_j es conocido, es fácil ver que $\alpha_{t+1}(j)$ se obtiene contabilizando para la observación O_{t+1} en el estado j, es decir, multiplicando la cantidad acumulada por la probabilidad $b_j(O_{t+1})$. El cálculo para la ecuación (1.4) se lleva a cabo para todos los estados j, donde $1 \leq i \leq N$, en un instante t dado; después el cálculo se realiza para $t = 1, 2, \ldots, T-1$. Finalmente, en el paso de *terminación* se obtiene el cálculo de $P(O|\lambda)$ deseado, sumando las variables terminales hacia adelante (*forward*) $\alpha_T(i)$. De acuerdo con la ecuación 1.2.

$$\alpha_T(i) = P(O_1O_2\cdots O_T, q_T = S_i|\lambda) \tag{1.2}$$

por tanto $P(O|\lambda)$ es sólo la suma de las $\alpha_T(i)$, es decir, para un instante t, el cálculo de la variable hacia adelante, $\alpha_t(i)$, se lleva a cabo para todos los estados.

Luego es iterada para $t = 1, 2, \ldots, T-1$. A esto se refiere como la etapa hacia adelante del algoritmo hacia adelante-hacia atrás. Con esto es suficiente para calcular $P(O|\lambda)$ y resolver el problema de evaluación.

Algoritmo 1.1 Algoritmo forward

Paso 1: *Inicialización*

$$\alpha_1(i) = \pi_i b_i(O_1) \quad 1 \leq i \leq N, \tag{1.3}$$

Paso 2: *Inducción*

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t-1}), \qquad \begin{array}{c} 1 \leq t \leq T, \\ 1 \leq j \leq N. \end{array} \tag{1.4}$$

Paso 3: *Terminación*

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{1.5}$$

Enseguida se tratará la parte hacia atrás (backward) del algoritmo hacia adelante-hacia atrás ya que es indispensable en la solución del problema de aprendizaje.
Como en el caso de la variable hacia adelante, $\alpha_t(i)$, ahora considérese la *variable hacia atrás* (*backward variable*), $\beta_t(i)$, definida como

$$\beta_t(i) = P(O_{t+1} O_{t+2} \cdots O_T | q_t = S_i, \lambda) \tag{1.6}$$

es decir, la probabilidad condicional de observar los vectores acústicos desde el instante $t+1$ hasta el instante T, estando en el estado S_i en el instante t, dado el modelo λ. En forma similar al cálculo de la variable hacia adelante, $\beta_t(i)$ puede ser calculada en forma recursiva usando el algoritmo backward que se representa en el algoritmo 1.2:

Algoritmo 1.2 Algoritmo backward

Paso 1: *Inicialización*

$$\beta_T(i) = 1, \quad 1 \leq j \leq N. \tag{1.7}$$

Paso 2: *Inducción*

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \qquad \begin{array}{c} 1 \leq t \leq T-1, \\ 1 \leq j \leq N. \end{array} \tag{1.8}$$

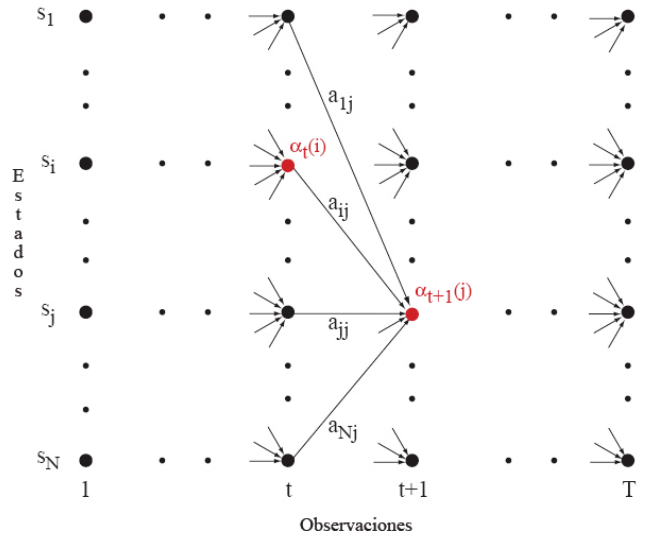


Figura 1.4: Etapa forward del algoritmo hacia adelante-hacia atrás [46].

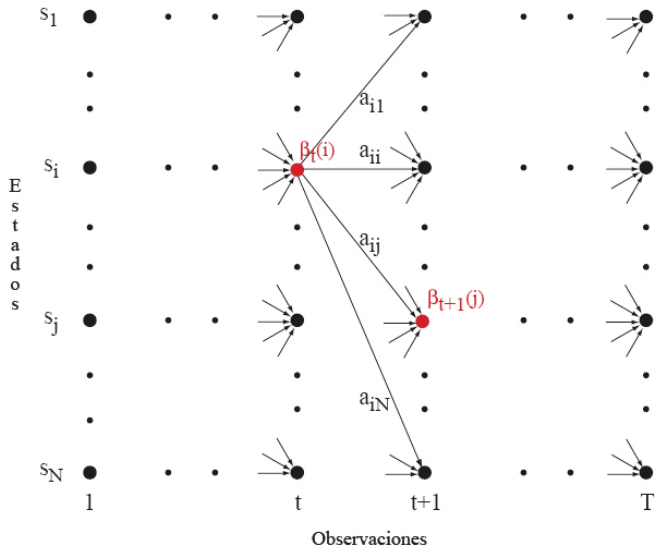


Figura 1.5: Etapa backward del algoritmo hacia adelante-hacia atrás [46].

En el paso de *inicialización* se establece $\beta_T(i)$ a 1 para todas las i. En el paso de *inducción*, el cual se ilustra en la figura 1.5, muestra que para estar en el estado S_i en el instante t y considerando la secuencia de observación en el instante $t+1$ hacia adelante, se toman todos los posibles estados S_j en el instante $t+1$, tomando en cuenta la transición del estado S_i al estado S_j (el término a_{ij}), así como la observación O_{t+1} en el estado j (el término $b_j(O_{t+1})$) y entonces se considera para la secuencia de observación parcial restante, desde el estado j (el término $\beta_{t+1}(i)$) [35].

Note que $\alpha_t(i)$ es la probabilidad conjunta de llegar al estado S_i en el instante t y observar los primeros t vectores acústicos, mientras que $\beta_t(i)$ es la probabilidad condicional de observar los últimos $T-t$ vectores dados estando en el estado S_i en el instante t. Esta asimetría permite el cálculo de la probabilidad de ocupación de un estado al tomar el producto de las variables hacia adelante y hacia atrás.

El problema de decodificación

La tarea de decodificación puede ser solucionada por medio del algoritmo de Viterbi, que cosiste en hacer coincidir una secuencia de vectores de observación no identificados, $O = O_1, O_2, \ldots, O_T$, con alguna de los modelos disponibles. Para encontrar la mejor secuencia de estados única para las primeras t observaciones es necesario definir $\delta_t(i)$ como el mejor indicador a través de una ruta, en el instante t, finalizando en el estado i.

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P(q_1, q_2, \ldots, q_{t-1}, q_t = i, O_1, O_2, \ldots, O_t|\lambda) \tag{1.9}$$

$\delta_t(i)$ puede ser calculado al usar la siguiente fórmula recursiva:

$$\delta_t(j) = [\max_{1 \le i \le N} \delta_{t-1}(i)a_{ij}]b_j(O_t), \quad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{1.10}$$

con las siguientes condiciones iniciales:

$$\delta_1(i) = \pi_i b_i(O_1), \quad 1 \le i \le N \tag{1.11}$$

La mejor secuencia de estados para los vectores de observación O se obtienen al mantener registro del argumento que fue maximizado por la ecuación (1.11) para cada t y cada j. Se puede hacer esto por medio del arreglo $\psi_t(j)$. El procedimiento completo para encontrar la mejor secuencia de estados se muestra en el algoritmo 1.3.

Se puede observar que el algoritmo de Viterbi es similar (excepto por la recuperación de la trayectoria) en la implementación del algoritmo *hacia adelante*. La mayor diferencia radica en que, el algoritmo de Viterbi utiliza una maximización en el paso de recursión (ecuación. (1.14)), mientras que el algoritmo *hacia adelante* utiliza una sumatoria (ecuación (1.4)). También es claro que la estructura de celosía (*lattice*) se implementa eficientemente con el procedimiento de Viterbi [30].

Algoritmo 1.3 Algoritmo de Viterbi

Paso 1: *Inicialización*

$$\delta_1(i) = \pi_i b_i(O_1), \qquad \text{Para } i = 1, 2, \cdots, N \tag{1.12}$$

$$\psi_1(i) = 0. \tag{1.13}$$

Paso 2: *Recursión*

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \quad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{1.14}$$

$$\psi_t(j) = \arg \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \quad \begin{matrix} 2 \le t \le T \\ 1 \le j \le N \end{matrix} \tag{1.15}$$

Paso 3: *Terminación*

$$p* = \max_{1 \le i \le N} [\delta_T(i)] \tag{1.16}$$

$$q_T* = \arg \max_{1 \le i \le N} [\delta_T(i)]. \tag{1.17}$$

Paso 4: *Recuperación de la trayectoria (secuencia de estados)*

$$q_T* = \psi_{t+1}(q_{t+1}*), \qquad \text{Para } t = T-1, T-2, \ldots, 1. \tag{1.18}$$

En reconocimiento de voz continua, cuando se utilizan modelos estadísticos de lenguaje (N-gramas), el algoritmo de Viterbi gasta una cantidad de tiempo significativa en cada trama para evaluar las transiciones entre palabras. En el caso de reconocimiento de voz continua o de sistemas que manejen subpalabras, los HMM pueden ser concatenados y el algoritmo de Viterbi encontrará la mejor secuencia en el modelo que corresponda a los datos de observación.

El problema de aprendizaje

El problema de aprendizaje puede ser resuelto utilizando un procedimiento iterativo como el *algoritmo de Baum-Welch*, que es una instancia específica del algoritmo *Maximización de la Esperanza* (*EM*). En otras palabras, la combinación de las etapas hacia adelante y hacia atrás del algoritmo hacia adelante-hacia atrás se usa para resolver el problema de aprendizaje y obtener los parámetros de Máxima Similitud (Maximum Likelihood).
Para ajustar los parámetros del modelo λ con el fin de maximizar la probabilidad de la secuencia de observación dado el modelo, resulta útil definir a $\xi_t(i,j)$, como la probabilidad de estar en el estado i en el instante t, y en el estado j en el instante $t+1$, dados la secuencia de observación O y el modelo λ y la secuencia de observación, es decir:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{1.19}$$

$\xi_t(i,j)$ también puede escribirse utilizando la definición de las variables *hacia adelante* (*forward*) y *hacia atrás* (*backward*) como:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} \tag{1.20}$$

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{1.21}$$

donde $P(O|\lambda)$ es un factor de normalización para obtener la probabilidad deseada. La secuencia de eventos que conducen a las condiciones requeridas por la ecuación (1.21) se ilustran en la figura 1.6.

Luego, se define $\gamma_t(i)$ como la probabilidad de estar en el estado s_i en el intante t, dados la secuencia de observación y el modelo; por tanto se puede relacionar $\gamma_t(i)$ y $\xi_t(i,j)$ al sumar sobre j obteniendo:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{1.22}$$

Al sumar $\gamma_t(i)$ sobre el índice de tiempo t, se obtiene una cantidad que puede ser interpretada como el número de veces que el estado s_i es visitado, o

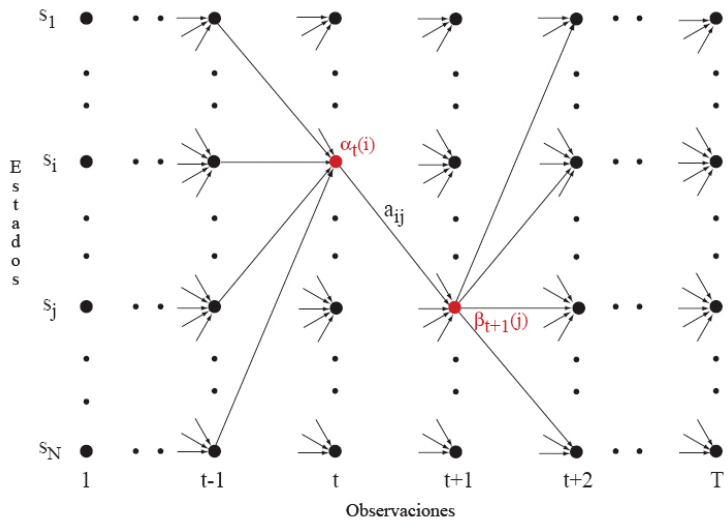


Figura 1.6: Operaciones requeridas para el cálculo de $\xi_t(i,j)$, es decir, la probabilidad de estar en el instante t en el estado i y en el instante $t+1$ en el estado j [46].

equivalentemente, el número esperado de transiciones hechas a partir del estado s_i, si se excluye el instante $t = T$ de la sumatoria. En forma similar, la sumatoria de $\xi_t(i,j)$ sobre t (desde $t = 1$ hasta $t = T-1$) puede ser interpretada como el número esperado de transiciones desde el estado s_i al estado s_j. Esto se expresa en las ecuaciones (1.23) y (1.24).

$$\sum_{t=1}^{T-1} \gamma_t(i) = \texttt{número esperado de transiciones hechas a partir del estado } \texttt{s}_\texttt{i} \tag{1.23}$$

$$\sum_{t=1}^{T} \xi_t(i,j) = \texttt{número esperado de transiciones desde el estado } \texttt{s}_\texttt{i} \texttt{ al estado } \texttt{s}_\texttt{j} \tag{1.24}$$

Usando las fórmulas de las ecuaciones (1.23) y (1.24), y el concepto de conteo de ocurrencias de eventos, podemos dar un método para la reestimación de parámetros de un HMM. Un conjunto razonable de fórmulas para π, A y B se presenta en (1.25), (1.26) y (1.27).

$$\begin{aligned}
\bar{\pi} &= \begin{array}{l}\texttt{número esperado de veces en}\\ \texttt{el estado } \texttt{s}_\texttt{i} \texttt{ en el instante t} = 1\end{array} \\
&= \gamma_1(i) \qquad (1.25)
\end{aligned}$$

$$\begin{aligned}
\bar{a}_{ij} &= \frac{\begin{array}{l}\texttt{número esperado de transiciones}\\ \texttt{desde el estado } \texttt{s}_\texttt{i} \texttt{ al estado } \texttt{s}_\texttt{j}\end{array}}{\begin{array}{l}\texttt{número esperado de transiciones}\\ \texttt{hechas a partir del estado } \texttt{s}_\texttt{i}\end{array}} \\
&= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i,j)} \qquad (1.26)
\end{aligned}$$

$$\begin{aligned}
\bar{b}_j(k) &= \frac{\begin{array}{l}\texttt{número esperado de veces en el estado j}\\ \texttt{y observando el símbolo } \texttt{v}_\texttt{k}\end{array}}{\begin{array}{l}\texttt{número esperado de veces}\\ \texttt{en el estado j}\end{array}} \\
&= \frac{\sum_{\substack{t=1 \\ O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \qquad (1.27)
\end{aligned}$$

Las ecuaciones (1.25), (1.26) y (1.27) son conocidas como las fórmulas de reestimación de Baum-Welch. Cada reestimación garantiza incrementar $P(O|\lambda)$, a menos que se haya alcanzado algún punto crítico, en cuyo caso la reestimación permanecerá sin cambio o decrecerá si existe un sobreentrenamiento. En otras palabras, si $\lambda = (A, B, \pi)$ es el modelo inicial y $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ es el modelo re-estimado se puede probar que [46]:

1. El modelo inicial, λ, define un punto crítico de la función de probabilidad, en cuyo caso $\bar{\lambda} = \lambda$ o

2. El modelo $\bar{\lambda}$ es más probable que λ en el sentido de que $P(O|\bar{\lambda}) > P(O|\lambda)$, es decir, que se encuentra un nuevo modelo $\bar{\lambda}$ a partir del cuál es más probable que se produzca la secuencia de observación.

Así podemos mejorar la probabilidad de la secuencia O que está siendo observada a partir del modelo si se usa $\bar{\lambda}$ iterativamente en lugar de λ y se repite la reestimación hasta que algún punto límite haya sido alcanzado. El modelo resultante es denominado *HMM de máxima similitud*. Se debe señalar que el algoritmo hacia adelante-hacia atrás conduce a un máximo local solamente y

en la mayoría de los problemas de interés, la superficie de optimización es muy compleja y tiene muchos máximos locales.
Las fórmulas de reestimación (1.25), (1.26) y (1.27) pueden ser derivadas directamente al maximizar la función auxiliar de Baum (utilizando técnicas estándar de optimización restringidas) [35]:

$$Q(\lambda|\bar{\lambda}) = \sum_{Q} P(Q|O,\lambda) \log[P(O,\ Q|\bar{\lambda}] \tag{1.28}$$

sobre $\bar{\lambda}$. La maximización de $Q(\lambda|\bar{\lambda})$ conduce al incremento de la probabilidad, es decir,

$$\arg\ \underset{\bar{\lambda}}{\text{máx}}[Q(\lambda|\bar{\lambda})] \Rightarrow P(O|\bar{\lambda}) \geq P(O|\lambda) \tag{1.29}$$

Eventualmente la función de probabilidad converge a un punto crítico.

1.2.2. Variaciones de los modelos ocultos de Markov

Mezclas Gaussianas

Al asumir ciertas propiedades de los vectores continuos multidimensionales, es posible estimar los parámetros de salida a partir de los datos de entrenamiento. Uno de las distribuciones de probabilidad continua utilizadas frecuentemente es la *mezcla Gaussiana* de M-componentes, como se observa en la figura 1.7.

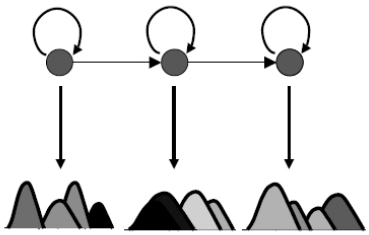


Figura 1.7: HMM continuo con mezclas Gaussianas [46].

Los HMM fueron introducidos en ASR al considerar a los vectores de observación como un conjunto de símbolos discretos que pertenece a un alfabeto finito, en cuyo caso es posible utilizar una función de densidad de probabilidad discreta dentro de cada estado del modelo. Puesto que los vectores de observación frecuentemente son de naturaleza continua, se han utilizado varios métodos para transformar señales continuas en secuencias de símbolos discretos; en particular los métodos de Cuantización Vectorial que se basan en técnicas de agrupamiento como los algoritmos K-medias y LBG. Sin embargo, al pasar una señal continua a una señal discreta se generan distorsiones que pueden ocasionar, en algunos casos, que se reduzca significativamente el desempeño del reconocedor.

Para superar las limitaciones de los HMM discretos, se han propuesto HMM de densidad de probabilidad continua (CHMM). En tal caso, algunas restricciones deben ser impuestas en las funciones de densidad de probabilidad, $\bar{b}_j(O)$. El modelo general de funciones de densidad de probabilidad para que se realice una reestimación más consistente ha sido, una mezcla finita de M funciones Gaussianas, como se observa en la ecuación (1.30):

$$\bar{b}_j(O) = \sum_{k=1}^{M} c_{jk} N(O, \mu_{jk}, \Sigma_{jk}) \quad 1 \leq j \leq N \tag{1.30}$$

donde O es el vector de observaciones para ser modelado, c_{jk} es el coeficiente de la mezcla para el k-ésimo componente de la mezcla, N es una densidad Gaussiana con vector de medias μ_{jk} y matriz de covariancias Σ_{jk} asociadas con el estado j y la mezcla k y debe de cumplir con:

$$c_{jk} \geq 0 \quad (1 \leq j \leq N \;\; 1 \leq k \leq M \; y \sum_{k=1}^{M} c_{jk} = 1, \; 1 \leq j \leq N) \tag{1.31}$$

En un CHMM la distribución de probabilidad del símbolo de salida se estima en forma similar al caso discreto haciendo uso del procedimiento hacia adelante-hacia atrás o con el algoritmo de Viterbi.
En los diferentes tipos de estructuras de los HMM que han sido estudiados cabe mencionar dos variaciones:

- La primera consiste de asociar a las observaciones con arcos en lugar de asociarlos con los estados de un HMM. Este tipo de modelo se ha encontrado que es bastante útil para definir las *transiciones nulas*, es decir, transiciones que no producen símbolo de observación o salida [46]. Tales transiciones proporcionan una manera muy eficiente de describir eliminación de fonemas en las alternativas de pronunciación de una palabra. En la figura 1.8 se observa que un modelo de Markov también se puede representar con una celosía (trellis) y que las transiciones nulas se representan con línea punteada.

- La segunda es el concepto de ligado de parámetros. La idea básica es reducir el número de parámetros HMM que sean independientes al configurar una equivalencia entre parámetros de diferentes estados. Este método resulta bastante interesante cuando la cantidad de datos de voz que se necesitan para el entrenamiento es insuficiente, puesto que el ligado de parámetros reduce el tamaño del HMM. Así, un modelo con estados ligados frecuentemente es más robusto que un modelo clásico entrenado con el mismo conjunto de datos, pero el modelo clásico será más preciso, proporcionando que los datos de entrenamiento sean suficientes. Es importante mencionar que las matemáticas del algoritmo de entrenamiento no se ven afectadas al ligar los parámetros HMM. Este punto se tratará con más detalle en el capítulo 2.

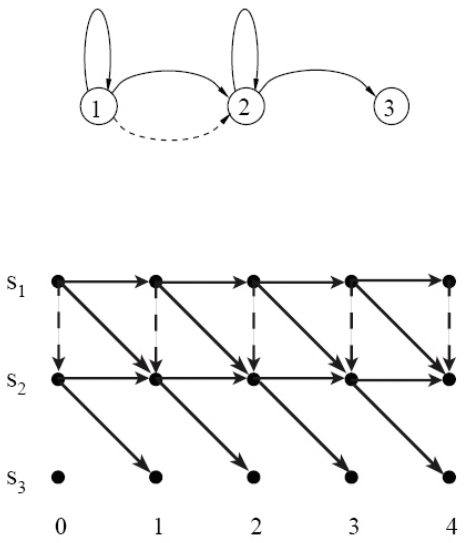


Figura 1.8: Representación de HMM mediante una celosía y uso de transiciones nulas.

Modelado de la duración de las palabras

Otro tópico importante basado en el estudio de los HMM es el *modelado de la duración de estados*. La geometría implícita "exponencialmente decreciente" de la densidad de la duración de estados, para estados con ciclos, como se observa en la ecuación (1.32) y donde $P_i(d)$ es la probabilidad de tomar el mismo ciclo en el estado i durante d veces, es inherente a un HMM, en su mayoría resulta inapropiado para la representación de una señal de voz. Lo que requiere de un modelado de densidad de la duración que sea explícito.

$$P_i(d) = (1 - a_{ii})a_{ii}^d \tag{1.32}$$

Aún cuando el marco teórico para incorporar la información de la duración de estado en un HMM se encuentra bien dominado, generalmente, su costo computacional es muy eleveado. Así, varios métodos han sido propuestos, tal es el caso del método de estimación de la distribución de probabilidad de la densidad de la duración de estado en forma directa sobre los datos de entrenamiento. Estas distribuciones de probabilidad pueden ser usados para un postprocesador heurísticamente pesando los indicadores obtenidos por la decodificación de Viterbi.

En el sistema *Sphinx* se incorpora el modelo de duración de las palabras como parte de la búsqueda de Viterbi. La duración de una palabra se modela por una distribución Gaussiana univariable, con media y variancia estimadas a partir de la segmentación de Viterbi supervisada de un conjunto de entrenamiento. Al

calcular previamente el record (marcador) de duración para varias duraciones, este modelo de duración esencialemente no se sobrepasa [21].

1.2.3. Entrenamiento

El procedimiento de entrenamiento para un reconocedor de voz basado en HMM comprende varios pasos tal como la construcción del modelo inicial, el modelo de reestimación supervisada y posiblemente entrenamiento correctivo. Durante la fase de construcción del modelo inicial, se determinan los estimados iniciales de los parámetros del HMM. Esta fase es muy importante, puesto que el proceso de entrenamiento iterativo encuentra solamente óptimos locales y, por tanto, es suceptible a iniciales. No existen métodos sencillos para escoger un buen modelo inicial. Las soluciones que se proponen incluyen segmentación de observaciones, ya sea en por medio de un criterio de máxima similitud o en forma manual, obteniendo promedios dentro de los estados, o utilizando segmentación *K-medias* con agrupamiento. En los HMM discretos, si la probabilidad se inicializa con cero, permanecerá cero siempre. De esta manera, es importante tener un conjunto de estimados iniciales que sea razohable. Estudios empíricos han demostrado que para HMM discretos se puede utillzar una distribución uniforme como estimado inicial. Funciona razonablemente bien en la mayoría de las aplicaciones; aunque los buenos estimados son siempre útiles para obtener las probabilidades de salida.
Si se utilizan densidades de mezclas continuas, son esenciales buenos estimados iniciales para las funciones de densidad de probabilidad Gaussianas. Existen varias maneras de obtener tales estimados:

- Se puede hacer uso del algoritmo de agrupamiento *K-medias*, como se utiliza en cuantización vectorial (*Vectorial Quantization. VQ*). La segmentación de estados de Markov puede ser derivada a partir de los HMM discretos, puesto que este procedimiento no es sensible a los parámetros iniciales. Basados en los datos de segmentación, se puede utilizar el algoritmo k-medias para derivar los parámetros de las medias y covariancias Gaussianas necesarias. Los coeficientes de las mezclas pueden tener distribución uniforme.

- Se pueden estimar los HMM semi-continuos a partir de los HMM discretos. Simplemente es necesario estimar una matriz de covariacias adicional para cada palabra código VQ y correr cuatro o cinco iteraciones adicionales para refinar los HMM semi-continuos basados en los HMM discretos, que típicamente requieren de cuatro o cinco iteraciones a partir de la distribución uniforme. Cuando los HMM semi-continuos son entrenados, se toman las M palabras código más relevantes y cada estado de Markov los usa como funciones de densidad Gaussiana iniciales para modelar las mezclas de densidad Gaussiana continuas.

- Se puede empezar entrenando un solo modelo de mezcla de densidad Gaussiana. Se pueden calcular los parámetros a partir de datos previamente

segmentados. Se pueden dividir iterativamente las funciones de densidad Gaussiana en forma similar a la generación de un libro código de VQ. Típicamente se requieren de dos o tres iteraciones para refinar las densidades continuas después de cada división.

La fase de reestimación supervisada del modelo corresponde al problema de aprendizaje mencionado en la sección 1.2.1. En el caso de voz continua se acepta la reestimación incrustada (imbuida). El entrenamiento embebido utiliza el mismo algoritmo de Baum-Welch como si se tratase del caso para palabras aisladas, pero en lugar de entrenar cada modelo individualmente se entrenan todos los modelos en paralelo. Durante esta fase de entrenamiento, se utilizan técnicas de estimación por Máxima Similitud (MLE) para reestimar los parámetros del modelo. Como se vió en la sección anterior la estimación MLE estándar se encarga de obtenter el conjunto de parámetros del HMM al maximizar $P(O^i|\lambda_i)$. O^i representa una secuencia de entrenamiento de las observaciones de voz usadas para derivar el conjunto de parámetros para el HMM λ_i. Además del criterio MLE, también se han propuesto criterios alternativos, como el *criterio de Información Mutua Máxima* (*Maximimum Mutual Information, MMI*) y el *criterio de Discriminación Mínima de Información* (*Minimum Discrimination Information, MDI*).
La estimación por el criterio de Información Mutua Máxima (MMIE) se basa en la idea de que diseñando todos los HMM al mismo tiempo la potencia de discriminación de cada modelo puede ser mejorada. Si suponemos que las unidades de voz representadas por los HMM son equiprobables, MMIE estima el conjunto de parámetros del HMM al maximizar:

$$I_i = \max_{\lambda} \left(\log P(O^i|\lambda_i) \; - \; \log \sum_{j=1}^{M} P(O^i|\lambda_j) \right) \tag{1.33}$$

donde M representa el número de HMM. Si comparamos esta fórmula con la de MLE, la suma de un segundo término permite al procedimiento de entrenamiento elegir λ para mejorar la separación entre el modelo correcto λ_i y todos los otros modelos en la secuencia de entrenamiento O^i. Cuando se aplica a todas las secuencias de entrenamiento, esta maximización se convierte en:

$$I = \max_{\lambda} \left(\sum_{i=1}^{M} \left(\log P(O^i|\lambda_i) \; - \; \log \sum_{j=1}^{M} P(O^i|\lambda_j) \right) \right) \tag{1.34}$$

Para resolver el problema de reestimación, se usa el algoritmo Baum-Welch extendido al caso MMIE. Comparado con el criterio MLE, MMIE puede mejorar el desempeño de la tarea de reconocimiento a expensas de incrementar la complejidad computacional.
Otra alternativa para mejorar el procedimiento de entrenamiento es escoger los parámetros HMM que minimicen la información de discriminación entre una distribución de probabilidad del conjunto de observaciones y la del HMM. La

información de discriminación entre dos distribuciones de probabilidad P y Q con funciones de densidad de probabilidad p y q puede ser expresada con la siguiente ecuación:

$$D(Q||P) = \int q(y) \ln \left(\frac{q(y)}{p(y)} \right) dy \tag{1.35}$$

La idea detrás del entrenamiento con el criterio MDI es compensar la falta de coincidencia entre las mediciones y el modelo. El procedimiento comienza con un estimado del HMM mediante la aproximación MLE. Luego, para una HMM dado, la distribución de probabilidad de la fuente (conjunto de observaciones) es estimada minimizando la información de discriminación sobre todas las distribuciones de probabilidad de la fuente que son consistentes con las mediciones. Finalmente, dada una distribución de probabilidad de la fuente, un HMM, que minimiza la información de discriminación sobre todos los HMM, es estimado. Para llevar a cabo la reestimación se propuso un modelo generalizado del algoritmo de Baum-Welch.
Finalmente, puede ser introducida una fase de postprocesmiento, como el entrenamiento correctivo, para mejorar el desmpeño del reconocimiento al mejorar la potencia de discriminación de los modelos obtenidos durante la fase de reestimación. El procedimiento de entrenamiento correctivo se enfoca en aquellas porciones del HMM que sean las más importantes para discriminar entre unidades similares que sean sometidas al reconocimiento. La ventaja del entrenamiento correctivo es que consiste de un paso adicional si se compara con el criterio MLE, por lo que el procedimiento de entrenamiento no necesita ser reformulado.

1.2.4. Reconocimiento de voz usando modelos ocultos de Markov

El uso típico de modelos ocultos de Markov en reconocimiento de voz no es muy diferente del paradigma tradicional de reconocimiento de patrones. La aplicación exitosa de los métodos de los HMM generalmente involucra los siguientes pasos:

1. Define un conjunto de L clases de sonidos para ser modelados, por ejemplo, fonemas o palabras. Nombra a las clases $V = v_1, v_2, \ldots, v_L$;

2. para cada clase reune un conjunto de entrenamiento de tamaño considerable formado por locuciones etiquetadas que sean conocidas y que estén en la clase;

3. basados en cada conjunto de entrenamiento, hallar la solución del problema de estimación para obtener el mejor modelo λ_i para cada clase v_i, $i = 1, 2, \ldots, L$;

4. durante el reconocimiento evalua $P(O|\lambda_i)$, $i = 1, 2, \ldots, L$ para la locución O desconocida e identifica la voz que produjo O como perteneciente a la

clase v_j si

$$P(O|\lambda_j) = \max_{1 \leq i \leq L} P(O|\lambda_i) \tag{1.36}$$

2

Modelado acústico

Después de años de investigación y desarrollo, la precisión en el ASR permanece como uno de los retos más importantes en la investigación. Varios factores determinan la precisión, entre los que destacan, variaciones en el contexto, en el locutor y en el ambiente. El modelado acústico juega un papel crítico para obtener una mejor precisión y es la parte central de cualquier sistema de reconocimiento de voz.
Para la observación acústica $\mathbf{X} = X_1, X_2, \ldots, X_n$ dada, la meta del reconocimiento de voz es encontrar la secuencia correspondiente de palabras $\hat{\mathbf{W}} = w_1, w_2, \ldots, w_m$ tal que se obtenga la máxima probabilidad $P(\mathbf{W}|\mathbf{X})$ como se expresa en la ecuación (2.1).

$$\hat{\mathbf{W}} = \arg\max_{w} P(\mathbf{W}|\mathbf{X}) = \arg\max_{w} \frac{P(\mathbf{W})P(\mathbf{X}|\mathbf{W})}{P(X)} \tag{2.1}$$

Puesto que la maximización de la ecuación (2.1) se lleva a cabo con la observación de $\mathbf{X}$ fija, la maximización anterior es equivalente a la maximización de la siguiente ecuación:

$$\hat{\mathbf{W}} = \arg\max_{w} P(\mathbf{W})P(\mathbf{X}|\mathbf{W}) \tag{2.2}$$

El reto desde el punto de vista práctico consiste en cómo construir modelos acústicos precisos, $P(\mathbf{X}|\mathbf{W})$, y también modelos del lenguaje, $P(\mathbf{W})$, que realmente reflejen al lenguaje hablado que se desea reconocer. Para reconocimiento de voz de sistemas de vocabularios muy grandes es necesario descomponer una palabra en una secuencia de subpalabras. Así $P(\mathbf{X}|\mathbf{W})$ se relaciona estrechamente con el modelado fonético. $P(\mathbf{X}|\mathbf{W})$ debe tomar en cuenta las variaciones entre locutores, las variaciones de pronunciación, las variaciones ambientales y las variaciones de coarticulación de fonemas dependientes del contexto. Por último, pero sin restarle importancia, cualquier acústica estática o modelo del lenguaje no reunirá las necesidades de las aplicaciones reales. Así que es de gran importancia adaptar dinámicamente tanto $P(\mathbf{W})$ como $P(\mathbf{X}|\mathbf{W})$ para maximizar $P(\mathbf{W}|\mathbf{X})$ mientras se trabaje con sistemas

de lenguaje hablado. El proceso de decodificación para encontrar la mejor secuencia de palabras **W** que coincida con la señal de entrada **X** en sistemas de reconocimiento de voz es más que un problema de reconocimiento de patrones simple, debido a que en reconocimiento de voz continua se tiene un número infinito de patrones de palabras para comparar, como se verá con más detalle en el capítulo 4.
La teoría de los HMM es la base que sustenta a la teoría del modelado fonético acústico. Proporciona una herramienta poderosa para integrar segmentación, el ajuste en el tiempo, la coincidencia de patrones y el conocimiento del contexto de manera unificada.

2.1. Modelos de subpalabras

La cuestión principal acerca de cuál es la unidad de voz de mejor tamaño para un sistema ASR conduce a explorar unidades más pequeñas que una palabra y más grandes que un fono, para capturar los efectos de coarticulación. Por muchas razones la precisión de un sistema ASR se deteriora con unidades muy pequeñas. Para sistemas de reconocimeinto de gran vocabulario los difonos y los modelos de fonos ligados representan una compensación razonable evitando tanto la gran cantidad de memoria como la falta de precisión con modelos independientes del contexto.
Cuando se utilizan subpalabras como unidades de voz, la memoria de referencia debe contener, también, un *diccionario del lenguaje*, que relacione cada palabra con su secuencia de estas unidades, por ejemplo, la secuencia de difonos que constituyen cada palabra del vocabulario.
Al reemplazar los modelos de palabras con modelos de subpalabras se reduce la precisión del sistema ASR. Las principales dificultades son:

1. que los efectos de coarticulación y de acentuación se extienden más allá de los fonos adyacentes inmediatos y

2. construir modelos de palabras al concatenar unidades más pequeñas requiere de modificaciones significativas a las unidades mezcladas, como por ejemplo, en cada límite de cada unidad, plantillas de tramas de aplanamiento o mezclado de los estados de los HMM.

Además del aplanamiento espectral, algunas unidades deben ser cortadas cuando formen modelos de varias sílabas, debido a que la duración de una palabra no se incrementa linealmente con el número de sílabas en una palabra.

2.2. Modelado del contexto

El contexto es muy importante en el reconocimiento de voz a múltiples niveles. En una escala de tiempo corto como sucede en la longitud promedio de un

fonema, las limitaciones en la tasa de cambio del tracto vocal causan confusión en las características acústicas, fenómeno que se conoce como coarticulación. El obtener los más altos niveles posibles es la meta del desempeño de un sistema de reconocimiento de voz, lo que significa hacer un uso eficiente de toda la información contextual.
La tecnología actual de los HMM utiliza una aproximación al problema de tipo top-down (descendente) al modelar el contexto fonético. La influencia contextual en tiempo corto de la coarticulación se maneja al crear un modelo para todos los contextos fonéticos que sean lo suficientemente distintos entre sí, lo que conduce a una compensación entre la creación de suficientes modelos que representen de manera adecuada la elocución y mantener suficientes ejemplos de entrenamiento por cada contexto de tal manera que los parámetros del modelo puedan ser bien estimados. Las técnicas de agrupamiento y aplanamiento pueden proporcionar un compromiso razonable para mejorar la precisión y el almacenamiento del modelo. Para simplificar la búsqueda en los sistemas ASR, muchos sistemas usan *modelos independientes del contexto* (*Context-independent models, CI*), donde cada unidad a modelar es entrenada independientemente de sus vecinos, ignorando sus efectos de coarticulación. Con pequeñas cantidades de datos de entrenamiento se minimizan los problemas de subentrenamiento, donde los parámetros del modelo son estimados póbremente debido a la insuficiencia de datos cuando se utliizan numerosos *modelos dependientes del contexto* (*context-dependent models, CD*).
Si se utilizan unidades dependientes del contexto, se puede mejorar la precisión del reconocimiento en forma significativa, considerando que existen suficientes datos de entrenamiento para estimar dichos parámetros dependientes del contexto. Los fonemas dependientes del contexto han sido ampliamente utilizados en sistemas de reconocimiento de grandes vocabularios, gracias a que mejoran la precisión y pueden ser más sencillos de entrenar. El contexto se refiere generalmente a los fonos que son vecinos inmediatos tanto a la izquierda como a la derecha del fono en cuestión.
Un *modelo de trifonemas* es un modelo fonético que toma en consideración a los fonos vecinos, tanto el del lado derecho como el del lado izquierdo. Si dos fonos tienen la misma identidad pero distintos contextos derecho e izquierdo, se consideran que son diferentes trifonemas. Se sabe que a las diferentes realizaciones de un fonemas se les denomina alófonos. Los trifonemas son ejemplos de alófonos.
Aunque los contextos derecho e izquierdo que se usan en los trifonemas son importantes, sólo son dos de los diversos factores importantes que contribuyen a la realización de un fonema. Los modelos de trifonemas son importantes porque capturan la mayoría de los efectos de coarticulación más importantes. Generalmente son mucho más consistentes que los modelos independientes del contexto. Sin embargo, es necesario balancear la capacidad de entrenamiento y la precisión con varias técnicas que permiten compartir parámetros.
Los modelos CD son típicamente *trifonemas* que representan N fonemas cada uno con N^2 modelos; por ejemplo, el fonema /p/ tiene modelos separados para cada combinación de todos los vecinos posibles que se encuentran a la derecha

y a la izquierda de él. Los modelos CD capturan la mayoría de los efectos de coarticulación, pero al costo de desarrollo y de búsqueda de dichos modelos es N^3, para cuando $N \approx 30 - 40$ fonemas. En muchos casos no se cuenta con recursos suficientes de memoria o de datos de entrenamiento para soportar un conjunto completo de tales modelos.

2.3. Ligado de modelos de subpalabras

Una manera importante para reducir el subentrenamiento es compartir parámetros ligados entre los modelos, usando los mismos valores en todos los modelos pertinentes a un contexto dado. Separarar los trifonemas de los modelos CD para todas las secuencias de tres fonemas resulta ineficiente, puesto que muchos fonemas tienen efectos coarticulatorios similares; por ejemplo, los efectos de los labiales $/b,\ p,\ m/$ en una vocal adyacente son muy similares. Los *modelos ligados* comparten los mismos valores de los parámetros para reducir el número total de datos de entrenamiento; los modelos CD son ligados donde se observa que sus contextos tienen los mismos efectos fonéticos, como se ve en la figura 2.1 . El ligado puede hacerse en forma automática, como en el caso de árboles de decisión conducidos por datos con estimación no supervisada, o guiados por propiedades lingüísticas, agrupando los contextos con etiquetas como labiales, velares, fricativas, etc. Cuando el ligado involucra funciones de distribución de probabilidad Gaussianas, a dichos modelos se les denomina HMM semi-continuos.

Cuando dos o más redes de modelos de palabras comparten secciones de subpalabras, o cuando sus parámetros son ligados, la memoria para los modelos de referencia y para el cálculo del sistema ASR se reduce. La técnica de compartir parámetros se puede aplicar en varios niveles: a nivel de palabras, de sílabas, de difonos, de fonos, o de tramas (fenones).

Aún con los estados ligados, el uso de mezclas Gaussianas continuas en los costos computacional y de uso de memoria de los HMM es bastante. Algunos sistemas ASR ligan parámetros, es decir, usando un solo conjunto de distribuciones de densidad de probabilidad Gaussiana (PDF); entonces, cada estado HMM es caracterizado únicamente con sus pesos de las mezclas, como se observa en la figura 2.1. Asumimos que unos cuantos cientos de PDFs son suficientes para modelar todo el espacio acústico. El problema de entrenamiento para cada estado entonces se reduce a estimar estos pesos, en lugar de encontrar todos los parámetros para las mezclas de PDF Gaussianas por separado; es especialmente costoso si cada vector de cada trama tiene docenas de parámetros y se utilizan matrices de covariancias completas (no diagonales). El modelado basado en trifonemas asume que cada trifonema del contexto es distinto. De hecho muchos fonemas tienen un efecto muy similar en sus fonemas vecinos. La posición de nuestros articuladores tiene un efecto importante en cómo se pronuncias las vocales vecinas. Por ejemplo, los fonemas oclusivos $/b/$ y $/p/$ tienen efectos similares en la vocal siguiente, como se ve en la figura 2.2 para las palabras bata y pata. Entonces, es deseable encontrar instancias que tengan efectos

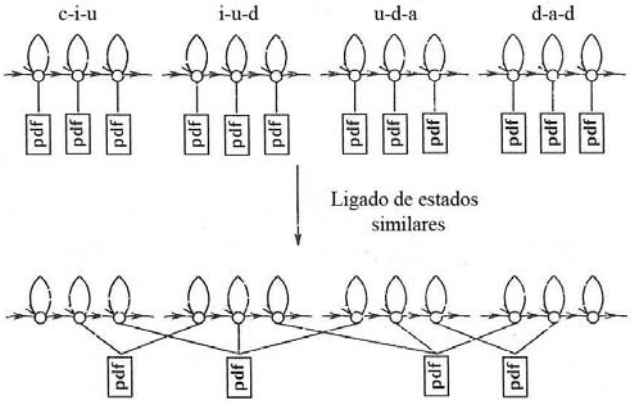


Figura 2.1: Ligado de estados para HMM [32].

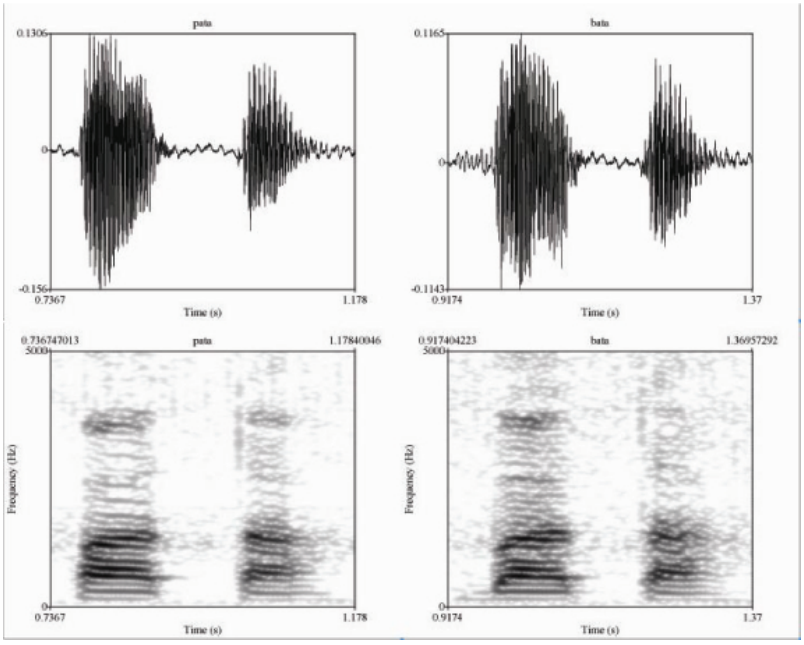


Figura 2.2: Efectos similares de fonemas en contextos diferentes.

similares y mezclarlas, lo que conduciría a un número menor de modelos que puedan ser manipulados y entrenados de mejor forma.
El balance entre la precisión y la capacidad de entrenamiento de los modelos de fonemas y los modelos de palabras pueden ser generalizados a modelar eventos subfonéticos. De hecho, las unidades fonéticas y subfonéticas presentan los mismos beneficios, conforme comparten parámetros a nivel de unidades. Esta es el beneficio clave en la comparación de unidades de palabras. Para el modelado subfonético se trata el estado en los HMM de subfonemas como unidades básicas. Hwang y Huang generalizaron el agrupamiento (clustering) de las distribuciones de probabilidad de salida de los estados dependientes por medio de modelos fonéticos diferentes [15]. Cada cluster representa, así, un conjunto de estados de Markov similares y es denominado *senón*. Después de que el agrupamiento ha finalizado cada modelo de subpalabra se descompone en una secuencia de senones. El número óptimo de senones para un sistema se determina principalmente por el corpus de entrenamiento disponible y puede ser ajustado sobre un conjunto de desarrollo. Cada modelo de alófono es un HMM formado de estados, transiciones y distribuciones de probabilidad. Para mejorar la confiabilidad de los parámetros estadísticos de estos modelos algunas distribuciones pueden ser ligadas. Por ejemplo, las distribuciones de la porción central de un alófono pueden ser ligadas para reflejar el hecho de que representan la parte estable (independiente del contexto) de la realización física de la parte central del fonema, pronunciado con la configuración estacionaria del tracto vocal. Agrupar la granularidad del estado en lugar de la granularidad del modelo puede conservar estados no similares de dos modelos aparte mientras que los estados similares son mezclados, conduciendo a que se compartan los parámetros de una mejor forma.

Existen dos temas importantes al crear unidades fonéticas o subfonéticas independientes del contexto:

- Es necesario permitir que los parámetros se compartan y suavicen de la mejor manera. Como ilustra la figura 2.2, muchos fonos tienen efectos similares en sus fonos vecinos. Si la realización acústica es claramente idéntica, los ligamos juntos para mejorar el entrenamiento y la eficiencia.

- Puesto que el número de trifonemas en español es muy grande (más de 100000), existen muchos trifonemas nuevos o que no has sido vistos que se encuentran en el conjunto de prueba pero no en el conjunto de entrenamiento. Es importante relacionar estos trifonemas que no has sido vistos con trifonemas entrenados apropiadamente.

Un *árbol de decisión* es un árbol binario para clasificar objetos deseados mediante la realización de preguntas binarias en forma jerárquica. Modelar trifonemas que no han sido vistos es particularmente importantes para lograr la independencia del vocabulario, ya que es difícil reunir un corpus de entrenamiento que cubra suficientes ocurrencias de cada posible unidad de subpalabra. Necesitamos encontrar modelos que sean precisos, entrenables y, especialmente, generaliza-

bles. El *árbol de decisión de senones* clasifica los estados de Markov de los trifonemas representados en el corpus de entrenamiento mediante la realización de preguntas lingüísticas formadas de conjunciones, disyunciones y/o de negaciones tomadas de un conjunto predeterminado de cuestiones lingüíticas divididas por categorías, como se observa en la figura 2.3. Como ejemplos de estas preguntas lingüísticas tenemos: ¿el fono del contexto izquierdo (L) es una fricativa? o ¿es el fono del contexto derecho (R) una vocal media?

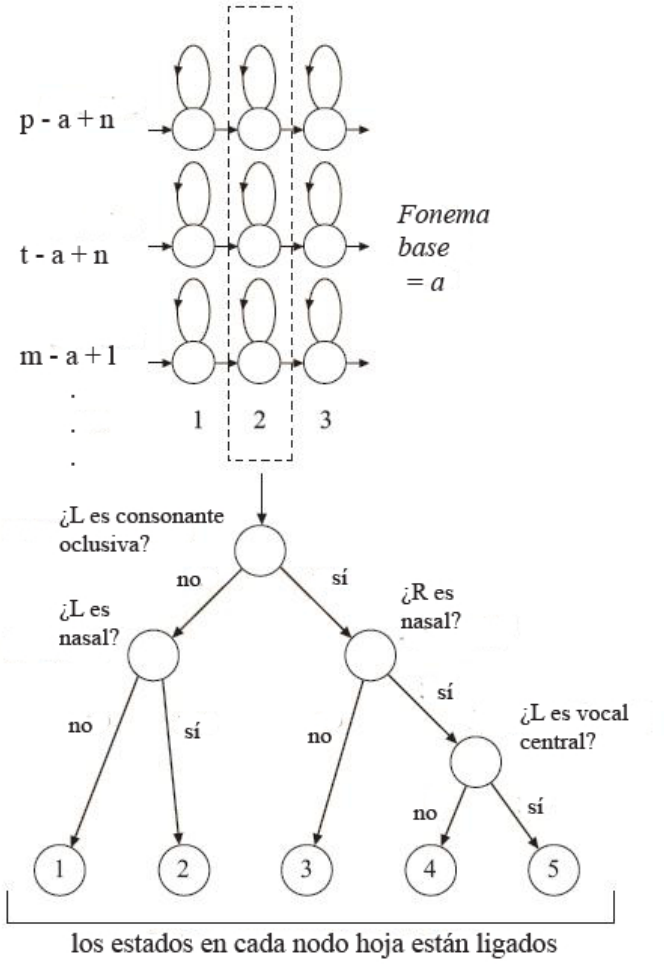


Figura 2.3: Árbol de decisión para establecer cuestiones lingüísticas [4].

A grandes rasgos la toma de decisiones consiste en que para cada nodo en el árbol revisamos si su fono derecho o izquierdo pertenece a una de las categorías. Se mide la correspondiente reducción de entropía o el incremento en la probabilidad para cada cuestión y para dividir el nodo se selecciona la pregunta que

posea el decrecimiento de entropía más grande. Así el árbol puede ser construido en forma automática al buscar, para cada nodo, la pregunta que presente el decrecimiento de máxima entropía. Alternativamente, se realizan preguntas más complejas para que cada nodo sea dividido de la mejor manera. Cuando el árbol crece, necesita ser podado usando validación de palabras cruzadas (cross-word validation). Cuando el algoritmo termina, las hojas del árbol representan los senones que serán usados.

2.4. Construcción de modelos de palabras

Si construimos una palabra HMM para cada palabra del vocabulario de un sistema de reconocimiento de palabras aisladas, el entrenamiento y la decodificación pueden ser implementados directamente usando los algoritmos básicos de la sección 1.2. Para estimar los parámetros del modelo, se deben recolectar ejemplos de las palabras del vocabulario. Los parámetros del modelo son estimados a partir de estos ejemplos mediante el algoritmo hacia adelante-hacia atrás y la fórmula de reestimación. No es necesario tener una detección del punto final precisa, porque el modelo del silencio automáticamente determina los límites si se concatenan modelos de silencio en ambos extremos de la palabra.
En sistemas de reconocimiento de gran vocabulario, si se utilizan unidades de subpalabras como modelos fonéticos, es necesario compartirlos con diferentes palabras. Estas unidades de subpalabras se concatenan para formar un modelo de palabra, posiblemente añadiendo modelos de silencio al principio y al final de la palabras, como se ilustra en la figura 2.4. Para formar un modelo de palabra a partir de modelos de subpalabras, se puede colocar una transición nula desde el estado final de la subpalabra HMM anterior al estado inicial de la siguiente subpalabra HMM, como se indica en la línea punteada de la figura 2.4.
En el ejemplo dado aquí, tenemos un fragmento del vocabulario empleado en el desarrollo de este sistema donde se incluyen también algunos de los fonemas del español hablado en México. El diccionario proporciona la información acerca de la pronunciación de cada palabra. Tenemos una palabra especial, $< SIL >$, que es el modelo del silencio, cuya correspondencia es /`sil`/ y cuya topolgía es la misma que la de los fonemas estándar. Para cada palabra en el vocabulario primero derivamos la secuencia fonética de ellas dada en en el diccionario. Se ligan estos modelos fonéticos para construir una palabra HMM. Procedimiento que se realiza para todas las palabras del vocabulario.

Por ejemplo para la palabra *EN* creamos un modelo de palabra basado en un silencio inicial, $/sil/$, seguido de los fonema $/E/$ y $/N/$, y terminando con otro silencio $/sil/$. El modelo compuesto se trata en la misma forma que cualquier otro HMM estándar. Se utiliza el algoritmo hacia adelante-hacia atrás para estimar los parámetros del modelo compuesto a partir de múltiples muestras de pronunciación de la palabra *EN*. Después de varias iteraciones, se obtienen los parámetros HMM en forma automática para $/sil/$, $/E/$ y $/N/$. Debido a que un fono puede encontrarse en distintas palabras, los parámetros fonéticos pueden ser estimados a partir de datos acústicos en diferentes palabras.

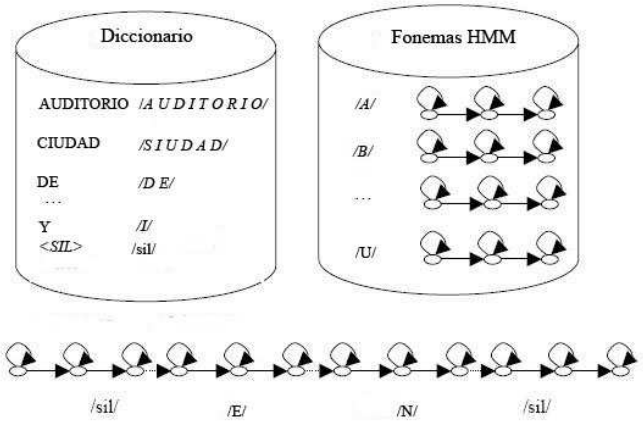


Figura 2.4: Construcción de un HMM por concatenación de subpalabras [15].

La habilidad para alinear en forma automática cada HMM individual con su correspondiente secuencia de observación no segmentada es una de las características más poderosas del algoritmo hacia adelante-hacia atrás. Cuando el método de concatenación de HMM se utiliza en voz continua, es necesario componer múltiples palabras para formar una oración de HMM basada en el archivo de la transcripción del discurso. De igual forma, el algoritmo hacia adelante-hacia atrás absorbe en forma automática un rango de información acerca de los posibles límites de las palabras, por lo que no es necesario tener una segmentación precisa en voz continua.

En general, para estimar los parámetros de los HMM, se crea una instancia de cada palabra con su modelo de palabra concatenado. Las palabras en la oración

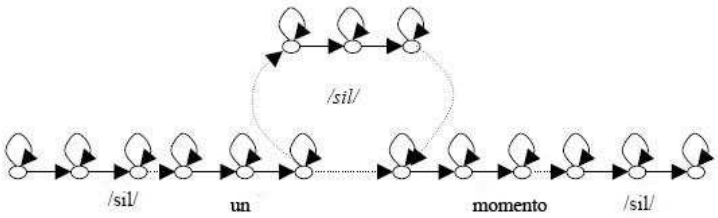


Figura 2.5: Construcción de un modelo de oración HMM por concatenación de palabras [15].

se concatenan con modelos de silencio opcionales entre ellas. En el ejemplo de la figura 2.5 se tiene un segmento de una oración utilizada en el presente trabajo. Se observa un HMM de silencio que está de manera opcional entre las palabras *un* y *momento*, y que se liga mediante una transición nula desde el

último estado del modelo de la palabra *un* hasta el primer estado del modelo de la palabra *momento*. Existe también una transición nula entre las palabras *un* y *momento* debido a que el silencio podría no existir durante el entrenamiento del ejemplo. Estas conexiones opcionales aseguran que se consideren todas las posibles realizaciones acústicas en voz continua, de tal manera que el algoritmo hacia adelante-hacia atrás pueda descubrir la ruta correcta en forma automática y estime con gran precisión el HMM para la secuencia de observación dada. Puesto que toda la oración HMM es entrenada con su correspondiente secuencia de observación, la mayoría de los límites de las palabras son inherentemente considerados. Los parámetros de cada modelo se basan en la alineación de los estados con la voz. No importa donde se encuentren los límites de las palabras.

3

Modelado del lenguaje

En los primeros días de los sistemas ASR únicamente se utilizaba información acústica para evaluar hipótesis de textos. Se asumía que la señal de voz tenía toda la información necesaria para convertir voz en texto. A principios de los años 80, se incorporó el conocimiento acerca del texto hablado y se encontró que mejoró significativamente la precisión del reconocedor al explotar la redundancia de texto. La mayoría de la voz corresponde a textos que siguen reglas lingüísticas, por ejemplo reglas de sintaxis y de semántica. La explotación de estas reglas es crucial en el desempeño de los sistemas ASR.

El propósito del modelo del lenguaje es proporcionar un mecanismo para la estimación de la probabilidad de alguna palabra w_k en una locución dada la secuencia de palabras $W_{k-1} = w_1, \ldots, w_{k-1}$ precedentes. La probabilidad a priori de una secuencia de palabras $W = w_1, \ldots, w_k$ en la ecuación (2.2) se puede obtener mediante la ecuación (3.1) y donde $P(\mathbf{W})$ corresponde al modelo del lenguaje y significa obtener la probabilidad de que w_k sea hablada dado que las palabras $w_1, \ldots w_{k-1}$ fueron pronunciadas previamente.

$$P(\mathbf{W}) = \prod_{k=1}^{K} P(w_k | w_{k-1}, \ldots, w_1) \tag{3.1}$$

La elección del modelo del lenguaje tiene un impacto significativo en el proceso de reconocimiento. En particular, la restricción que proporciona un modelo del lenguaje puede mejorar sustancialmente el desempeño del sistema y el tamaño del espacio de búsqueda generado por el modelo del lenguaje frecuentemente determina el algoritmo de búsqueda a utilizar. Los modelos de lenguaje que se usan en sistemas de reconocimiento de voz continua (CSR) caen en numerosas categorías:

- *Modelos de lenguaje uniformes*, donde cada palabra en cualquier oración se considera que es igualmente probable.
- *Modelos de lenguaje estocásticos*, tales como los modelos basados en unigramas, bigramas y trigramas, donde la probabilidad de una palabra en

una secuencia es estimada basándose en la probabilidad de que pertenezca a una clase determinada definida por las palabras que la preceden en esa secuencia. La probabilidad del lenguaje de la secuencia de palabras está dado por la probabilidad conjunta de todas las palabras en la secuencia. Una manera simple y efectiva de realizar esto es por medio de los N-gramas en los que se asume que w_k depende sólo se las $N-1$ palabras precedentes, esto es

$$P(\mathbf{W}) = \prod_{k=1}^{K} P(w_k|w_{k-1}, w_{k-2}, \ldots, w_{k-N+1}) \tag{3.2}$$

Los N-gramas codifican simultáneamente la partes sintáctica, semántica y pragmática y se concentran en las dependencias locales. Esto los hace muy efectivos para lenguajes como el español, en el que el orden de las palabras es importante y los efectos contextuales más fuertes tienden a aparecer cerca de sus vecinos cercanos.

- *Lenguajes de estados finitos*, no son más que lenguajes artificiales simples que modelan todas las oraciones legales utilizando una sola red. En los sistemas de reconocimiento de voz sencillos, los enunciados de entrada esperados generalmente son modelados con gramáticas en las que al usuario sólo se le permite pronunciar aquellos enunciados que están explícitamente cubiertos por la gramática, por ejemplo un reconocedor de dígitos, como se observa en la figura 3.1. que utiliza expresiones regulares en la definición de la gramática.
- Otras posibles categorías de modelos de lenguajes incluyen libres de contexto, unificación, basados en árboles estadísticamente y gramáticas de casos de marcos.

Estos modelos de lenguaje proporcionan varios niveles de restricción al reconocedor. Las gramáticas de estados finitos tienden a ser muy restrictivas, sin embargo, resultan ser gramáticas poco realistas para la mayoría de las tareas. Por otro lado, las gramáticas libres de contexto y las gramáticas de unificación resultan ser más realistas pero poseen demsasiados estados para enumerar. Por su parte, las gramáticas estadísticas pueden ser más pequeñas cuando se usan unigramas o bigramas pero cuando se usan trigramas el número de palabras previas que son modeladas incrementa, lo que ocasiona que el número de estados crezca rápidamente.

3.1. Aspectos prácticos del modelado del lenguaje

La descomposición secuencial de la ecuación (3.1) es apropiada para el reconocimiento de voz porque conduce al desarrollo de un criterio de decisión intermedio en forma natural que permite un retraso mínimo en la respuesta del reconocedor al dictado progresivo. En realidad, las probabilidades

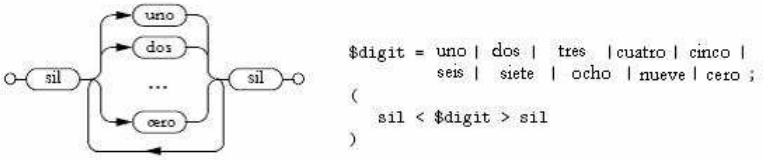


Figura 3.1: Gramática de estados finitos [33].

$P(w_k|w_1, \ldots w_{k-1})$ podrían llegar a ser imposibles de estimar aún para valores moderados de k: la mayoría de los historiales de secuencias de palabras $w_1, \ldots w_{k-1}$ podrían ser únicos o podrían ocurrir solamente unas cuantas veces. De hecho, para un vocabulario de tamaño L existen L^{k-1} historiales diferentes, así que para especificar $P(w_k|w_1, \ldots w_{k-1})$ completamente, los valores de L^k tendrían que ser estimados. Lo que se convierte en una cantidad astronómica para valores prácticos de L (para $L = 20000$ y $k = 3$, $L^k = 80$ `trillones`) y así las probabilidades $P(w_k|w_1, \ldots w_{k-1})$ no podrían ser ni almacenadas ni recuperadas cuando el reconocedor lo necesitara.
Un método bastante práctico de modelado del lenguaje de texto natural se basa en una simple clasificación de equivalencia: los historiales son equivalentes si terminan en las mismas dos palabras. Así

$$P(\mathbf{W}) = \prod_{k=1}^{n} P(w_k|w_{k-2}, w_{k-1}) \tag{3.3}$$

Lo que convierte la ecuación (3.3) en un modelo del lenguaje basado en trigramas. Lo más importante en este modelo es la estimación de las probabilidades básicas $P(w_3|w_1, w_2)$. Un primer impulso es hacerlo por la aproximación más simple que es la aproximación por frecuencias relativas:

$$P(w_3|w_1, w_2) = f(w_3|w_1, w_2) = \frac{C(w_1, w_2, w_3)}{C(w_1, w_2)} \tag{3.4}$$

donde la función C cuenta el número de veces que la cadena de su argumento ha ocurrido en el texto de entrenamiento. Analizando más detenidamente se verá que la ecuación (3.4) revela una fórmula inadecuada, puesto que muchos de los trigramas w_1, w_2, w_3 de las palabras posibles del español, no podrían ocurrir en corpora muy grandes. Un modelo del lenguaje basado en la ecuación (3.4) podría reafirmar el hecho de la imposibilidad de que existan tales trigramas, al asignar $P(W) = 0$ a las cadenas W que los contienen y podría conducir a un gran número de errores en el reconocedor que opera bajo el criterio de decisión estadística de la ecuación (2.2). Por lo tanto, es necesario "aplanar" o "suavizar" las frecuencias de los trigramas, lo que puede ser solucionado interpolando las frecuencias relativas de los unigramas, bigramas y trigramas mediante la ecuación:

$$P(w_3|w_1, w_2) = q_3 f(w_3|w_1, w_2) + q_2 f(w_3|w_2) + q_1 f(w_3) \tag{3.5}$$

donde pesos no negativos satisfacen la ecuación $q_1 + q_2 + q_3 = 1$. Puesto que $f(w_3|w_1, w_2)$ aproxima a $P(w_3|w_1, w_2)$ de mejor forma si se basa en una conteo muy grande de $C(w_1, w_2)$, la estimación será mejorada al permitir que los pesos dependan de las condiciones de conteo $C(w_1, w_2)$ y $C(w_2)$. Se trata, entonces, de elegir los valores de q_1 que satisfagan el criterio de máxima similitud para obtener una interpolación óptima.

3.2. Gramáticas en ASR

Sin un modelo del lenguaje, el sistema ASR generalmente genera una secuencia de símbolos que representan fonemas o segmentos fonéticos, que corresponden al texto reconocido propuesto. La secuencia puede involucrar conjuntos de posibilidades, por ejemplo, una lista pesada de localizaciones de límites y fonos candidatos para cada segmento, frecuentemente en la forma de una estructura *lattice*. Debido a la dificultad para localizar los límites de las palabras e incluso de las sílabas. La segmentación de la secuencia en palabras puede ser dejada a un postprocesador que compare la cadena de símbolos o la estructura lattice con el conjunto de palabras del vocabulario que particionan óptimamente la secuencia en palabras para el texto de salida. La *gramática* o estructura de secuencias de fonemas permitidos logra la precisión del sistema ASR al eliminar secuencias de fonos candidatos que no son permitidos por las reglas de la gramática. Las gramáticas desempeñan un rol muy importante en el reconocimiento de voz: Gobiernan la forma en la que se reconocerán las hipótesis de salida.
Las gramáticas pueden ser aplicadas de igual manera a secuencias de palabras a partir de la salida de un sistema de reconocimiento de palabras aisladas y de las secuencias de fonos a partir del análisis CSR. Tradicionalmente las gramáticas se refieren a reglas sintácticas por medio de las que las oraciones se analizan sintácticamente al descomponerlas en sus palabras y frases constitutivas. Debido a que el lenguaje natural tiene una gramática muy compleja, algunso sistemas CSR imponen restricciones a los usuarios de tal manera que las pruebas del discurso estén gobernadas por gramáticas más simples. Generalmente, las reglas de la gramática indican como las palabras con diferentes clases sintácticas pueden combinarse para formar oraciones válidas.
Los modelos de Markov son una forma de gramática de estados finitos, que no es lo suficientemente poderosa para generar todas las oraciones válidas del español. Las gramáticas *libres de contexto* y las gramáticas *sensibles al contexto* también han sido utilizadas en ASR y han sido generalizadas para incluir otros aspectos de la comunicación de la voz que tienen una estructura regular, por ejemplo en semántica y fonética.
El uso de HMM para describir la secuencia de segmentos acústicos en palabras a partir de un vocabulario es un ejemplo de gramática fonética. Las gramáticas para ASR se representan eficientemente por redes, donde cada red de estados representa un evento acústico y las transiciones entre estados denotan el orden permitido de eventos de acuerdo al vocabulario y a la gramática. Las redes de oraciones pueden manejar restricciones sintácticas y semánticas por medio de

una *red de transiciones aumentada*, que modifica el modelo estándar de Markov para permitir que la información lingüística sea obtenida y guardada en pilas durante cada transición.

3.3. Perplejidad

Considere cómo una gramática reduce la incertidumbre durante el reconocimiento. Sin una gramática, el vocabulario entero debe considerarse en cada punto de decisión. Con una gramática, es posible eliminar muchos candidatos desde la consideración, o asignarle una probabilidad más alta a algunos candidatos que a otros. La perplejidad es una medida de la restricción impuesta por la gramática, o del nivel de incertidumbre dado por la gramática [22].
Dado un modelo del lenguaje que asigna la probabilidad $P(\mathbf{W})$ a la secuencia de palabras $\mathbf{W}$, podemos derivar un algoritmo de compresión que codifique al texto $\mathbf{W}$ utilizando $-\log_2 P(\mathbf{W})$ bits. La entropía cruzada $H(\mathbf{W})$ de un modelo $P(w_k|w_{k-n+1}, \ldots, w_{k-1})$ en los datos $\mathbf{W}$, con una secuencia de palabras de suficiente longitud puede ser aproximada como:

$$H(\mathbf{W}) = \frac{1}{N_w} \log_2 P(\mathbf{W}) \tag{3.6}$$

donde N_w es la longitud del texto $\mathbf{W}$ medida en palabras. En términos matemáticos la perplejidad $PP(\mathbf{W})$ de un lenguaje $P(\mathbf{W})$ se define como el recíproco de la probabilidad promedio asignada por el modelo de cada palabra en el conjunto de prueba $\mathbf{W}$. La perplejidad es una medida que se relaciona con la entropía cruzada, conocida como *perplejidad de un conjunto de prueba* y está dada como:

$$PP(\mathbf{W}) = 2^{H(\mathbf{W})} \tag{3.7}$$

Así la perplejidad puede ser interpretada como la media geométrica del factor de ramificaciones del texto cuando es presentado al modelo del lenguaje. La perplejidad definida en la ecuación (3.7) tiene dos parámetros: un modelo del lenguaje y una secuencia de palabras. La perplejidad del conjunto de prueba evalua la capacidad de generalización del modelo del lenguaje. La *perplejidad del conjunto de entrenamiento* mide de qué manera el modelo del lenguaje se relaciona con los datos de entrenamiento, a manera de probabilidad. En general, una perplejidad baja está correlacionada con un mejor desempeño de reconocimiento. Esto se debe a que la perplejidad es una medida estadísticamente pesada de la ramificación de las palabras en un conjunto de prueba. Mientras más alta sea la perplejidad, habrá más ramas que el reconocedor tendrá que considerar en términos estadísticos.
El significado de que un lenguaje tenga un valor muy alto de perplejidad es que el número de ramificación de palabras a partir de la palabra anterior es en promedio más grande. En este sentido, la perplejidad es una indicación de la complejidad del lenguaje si tenemos una estimación precisa de $P(\mathbf{W})$. Para un lenguaje dado, la diferencia entre la perplejidad de un modelo del lenguaje y de la verdadera perplejidad del lenguaje es la indicación de la calidad del

modelo. La perplejidad de un determinado modelo de lenguaje puede cambiar dramáticamente en términos del tamaño del vocabulario, el número de estados o las reglas de la gramática y de las probabilidades estimadas.

3.4. Problemas con la estimación del modelo basado en trigramas

La fórmula de interpolación de la ecuación (3.5) que estima $P(w_3|w_1, w_2)$ presenta muchos problemas. Si el dictado contiene un trigrama w_1, w_2, w_3 que nunca tuvo lugar en el conjunto de entrenamiento, entonces w_3 debe ser estimada a partir de bigramas y unigramas. Si el texto de entrenamiento no contiene a w_2, w_3, entonces el modelo del lenguaje de la ecuación (3.5) recaerá en el cálculo de la frecuencia relativa del unigrama $f(w_3)$. Aquí la dificultad radica en que $f(w_3)$ centra la mayoría del peso en las palabras más frecuentes del vocabulario, tales como los artículos 'el', 'un', etc. Sin embargo, puesto que debe haber una buena razón lingüística por la que la secuencia $[w_2,\ '\texttt{el}']$ nunca ocurriría en el entrenamiento, el modelo del lenguaje debería asignar, de hecho, una probabilidad muy baja a la secuencia $[w_2,\ '\texttt{el}']$, más baja que las secuencias que no se han visto $[w_2,\ w_3'']$, donde w_3'' es una palabra de baja frecuencia .
Así la fórmula de la ecuación (3.5) podría ser mejorada al realizar un suavizado de los estimados de las frecuencias relativas $f(w_3|w_1, w_2)$ para compensar la dispersión de los datos en los que se basan. A estas técnicas se les denomina *estrategias de descuento*. El *descuento* es el proceso de reemplazar los conteos originales con conteos modificados, de tal manera que se redistribuya la masa de probabilidad a partir de los eventos más comúnmente observados hasta los eventos menos frecuentes y los eventos no vistos. Sea $c(E)$ el número actual de ocurrencias de un evento E, como la ocurrencia de un bigrama o de un trigrama. La cuenta modificada se define por la ecuación (3.8)

$$d(c(E))c(E) \tag{3.8}$$

donde $d(c(E))$ es el radio de descuento.

Los algoritmos de suavizado más comunes incluyen *back-off* o *deleted interpolation* y las estrategias de descuento que se utilizan comunmente son *Witten-Bell discounting* y *Good-Turing discounting*.

El algoritmo *Witten-Bell Discounting* se basa en una simple pero muy inteligente intuición acerca de eventos de frecuencia cero. Primero se piensa en una palabra o N-grama de frecuencia cero como un evento que no ha ocurrido todavía; cuando sucede, ésta tendrá que ser la primera vez que se observe dicho N-grama. Así, la probabilidad de ver un N-grama de frecuencia cero puede ser modelada mediante la probabilidad de observar un N-grama por primera vez. Éste es un concepto recurrente en procesamiento estadístico del lenguaje.
El calculo de la probabilidad de observar un N-grama por primera vez se realiza contando las veces que observamos un N-grama por primera vez en un corpus de entrenamiento. Esto es muy simple de producir debido a que la cuenta de los

N-gramas que se observa por primera vez es solo el número de tipos de N-gramas en los datos de entrenamiento. Se debe ver cada tipo de N-grama exactamente una vez [33].

En otras palabras, el radio de descuento no depende del conteo de eventos, sino del número de tipos t que le siguen a un contexto en particular. Se define al radio de descuento como

$$d(r,t) = \frac{n}{n+t} \tag{3.9}$$

donde n es la cantidad de palabras en el conjunto de entrenamiento. Esto es equivalente a establecer $P(w|h) = \frac{c}{n+t}$, donde w es una palabra, h es el historial y c es el número de ocurrencias de w en el contexto h, para eventos que ya han sido vistos y $P(w|h) = \frac{t}{n+t}$ para eventos que no han sido vistos.

El algoritmo de *Good-Turing Discounting* proporciona una forma un poco más compleja de realizar el suavizado. La idea básica del algoritmo es re-estimar la cantidad de masa de probabilidad para asignarla a los N-gramas con valor cero o muy bajo, observando el número de N-gramas con valores altos [33].

En el algoritmo Good-Turing el radio de descuento se define como

$$d(r) = \frac{(r+1)n(r+1)}{rn(r)} \tag{3.10}$$

donde $n(r)$ es el número de eventos que ocurren r veces. El descuento solamente se aplica a eventos que ocurren menos de K veces, donde K toma un valor de 7 típicamente.

4

Decodificación

4.1. Panorama del proceso de decodificación de voz continua

El reconocimiento de voz continua es a la vez una tarea de reconocimiento de patrones y un problema de búsqueda. A la acción de realizar una decisión de búsqueda en reconocimiento de voz se le denomina *decodificación* [15]. De hecho, la decodificación toma su nombre del modelo fuente-canal de la teoría de la información, que se observa en la figura 4.1.

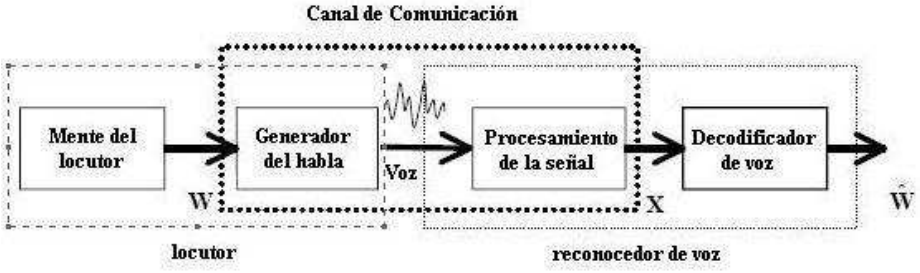


Figura 4.1: Modelo de comunicación fuente-canal [15].

La fuente de comunicación corresponde a la mente del locutor, quien especifica el conjunto de palabras **W** que serán pronunciadas por su aparato vocal, es decir, el generador del habla. La idea es obtener una señal **W** que corresponde al conjunto de palabras reconocidas por el decodificador. Para lograr esto, la voz ha sido transmitida por un canal ruidoso, formado por el generador del habla y el componente de procesamiento de la señal que es parte del reconocedor de voz. En el componente, también denominado procesador acústico, se obtiene la señal **X** que corresponde a la secuencia acústica de las palabras originales.

El decodificador, también llamado decodificador lingüístico [18], básicamente es un proceso de búsqueda para descubrir la secuencia de palabras $\hat{\mathbf{W}} = w_1, w_2, \ldots, w_m$ tal que se obtenga la máxima probabilidad $P(\mathbf{W}|\mathbf{X})$ para la observación acústica dada por $\mathbf{X} = X_1, X_2, \ldots, X_n$. Esto es, como ya se había comentado en el capítulo 2 y que se transcribe a continuación:

$$\hat{\mathbf{W}} = \arg \max_{w} P(\mathbf{W})P(\mathbf{X}|\mathbf{W}) \tag{4.1}$$

donde, como ya se sabe $P(\mathbf{W})$ representa el modelo del lenguaje, $P(\mathbf{X}|\mathbf{W})$ representa el modelo acústico, cada w_i representa a una palabra del vocabulario y cada X_j denota una observación acústica en un instante de tiempo determinado. La meta del proceso de reconocimiento del habla es encontrar una secuencia de palabras $\hat{\mathbf{W}}$ cuyos modelos acústico y de lenguaje correspondientes resulten ser la mejor coincidencia con la señal de entrada $\mathbf{W}$ [18]. Por tanto al proceso de decodificación junto con los modelos acústico y de lenguaje ya entrenados, se le conoce frecuentemente como *proceso de búsqueda* [15].

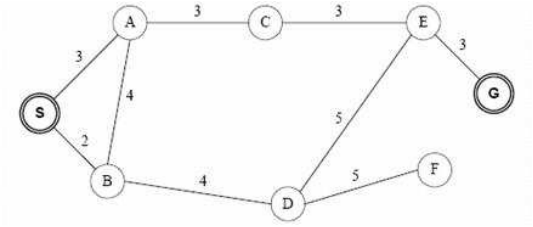
4.2. Reconocimiento de voz continua

La búsqueda en CSR se torna bastante complicada, aún cuando se trabaje con un vocabulario pequeño, ya que el algoritmo de búsqueda tiene que considerar la posibilidad de que cada palabra comience en cualquier trama de tiempo arbitraria. Debido a que no se conoce ni el número de palabras ni los límites de cada una de ellas o no se conocen los fonemas de la señal de entrada, se requiere de estrategias de búsqueda apropiadas para tratar con estos patrones no estacionarios de longitud variable. La complejidad de un algoritmo de búsqueda se encuentra altamente correlacionada con el espacio de búsqueda, el que a su vez se encuentra determinado por las restricciones que se imponen en el modelo del lenguaje.
La idea de búsqueda implica moverse alrededor, examinar cosas y tomar decisiones acerca de si el objeto objetivo ya sido alcanzado o aún no. En general, tales problemas pueden representarse usando el paradigma de *búsqueda por espacio de estados*. Tal paradigma se define como una tripleta (S, O, G), donde S es el conjunto de estados iniciales, O es un conjunto de operadores o reglas que se aplican en un estado para generar una transición, con su correspondiente costo, hacia otro estado, y G un conjunto de estados objetivo o metas. Una solución en el paradigma de búsqueda por espacio de estados consiste en hallar una ruta desde un estado inicial a un estado objetivo. El espacio de estados comúnmente se representa como un grafo dirigido en el que cada nodo corresponde a un estado y cada arco corresponde a la aplicación de un operador o regla, que permite una transición de un estado a otro. Así la búsqueda en el espacio de estados consiste en la búsqueda a través del grafo por medio de alguna función objetivo [15]. Otra medida importante para el grafo de búsqueda es el *factor de ramificación*, que se define como el número promedio de sucesores por cada nodo. Puesto que el número de nodos en un grafo de búsqueda crece

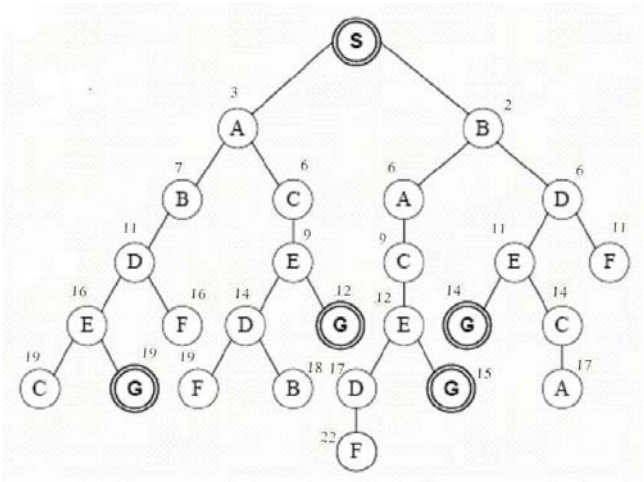
exponencialmente con base igual a este factor de ramificación.
El ejemplo más común es el problema del agente viajero que necesita hallar la ruta para viajar de la ciudad S a la ciudad G en la menor distancia posible, considerando que existen varias ciudades nombradas con letras mayúsculas y que el paso por cada una de ellas involucra una distancia o costo al pasar por ella, como se ve en la figura 4.2a. La solución al problema se da en la figura 4.2b donde se plantean las posibles alternativas a manera de un árbol de búsqueda, en donde cada nodo representa una ciudad y los valores junto a cada nodo representan los costos acumulados durante su viaje por cada ruta. Todas las técnicas de búsqueda utilizan dos estrategias: *compartir y podar* [15]. Por *compartir* se refiere a que los resultados intermedios pueden mantenerse, de tal manera que puedan ser usados por otras rutas sin tener que redundar en el cálculo cuando se alcancen otra vez. El *podado* significa que las rutas que no son prometedoras pueden ser descartadas sin perder tiempo en explorarlas más adelante.
Existen varias técnicas de búsqueda, sin embargo, en el reconocimiento de voz la decodificación generalmente se realiza con el algoritmo de Viterbi o con los decodificadores de pila A^*. La razón para elegir el decodificador de Viterbi es que involucra a los argumentos que señalan a la voz como un proceso que se realiza de izquierda a derecha y a la eficiencia que se logra por ser un proceso de sincronía de tiempo. Las razones para elegir un decodificador de pila involucran a la habilidad que tiene éste de explotar más eficientemente el criterio A^* es que conserva la esperanza de realizar una búsqueda óptima y de trabajar con espacios de búsqueda enormes. En este proyecto se trabaja con el algoritmo Viterbi beam ya que, gracias a las técnicas de podado, es el método más eficiente y más utilizado para casi todas las tareas de reconocimiento de voz. La búsqueda beam tiene el estilo breadth-fist (primero de lado a lado) y progresa también en profundidad. La diferencia con el método tradicional breadth-fist, la búsqueda beam sólo expande aquellos nodos cuya probabilidad de éxito es mayor en cada nivel y el resto es ignorado para mejorar la eficiencia. El ejemplo del agente resuelto por el método de beam se muestra en la figura 4.2c. Antes de profundizar con los detalles de la búsqueda beam es necesario manejar otros conceptos por lo que esta técnica se retomará en la sección 4.5.

4.3. Combinando los modelos acústico y de lenguaje

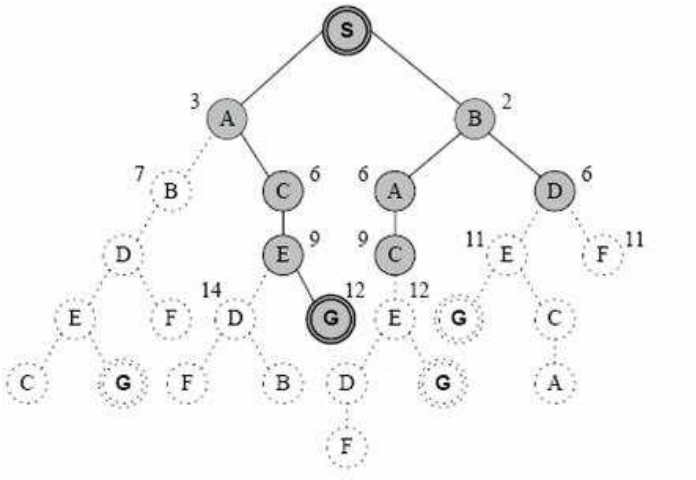
Aunque la ecuación de Bayes (ecuación (4.1)) sugiere que la probabilidad del modelo acústico (probabilidad condicional) y la probabilidad del modelo del lenguaje se combinen mediante una simple multiplicación, en la práctica, la asignación de algún peso es deseable. Por ejemplo, cuando los HMM son usados para el modelo acústico, la probabilidad acústica generalmente es subestimada, debido a la falacia de Markov y la suposición de independencia. Al combinar el modelo del lenguaje con una probabilidad subestimada del modelo acústico podría resultar en un modelo del lenguaje que tenga muy poca influencia en



(a)



(b)



(c)

Figura 4.2: a) Problema del agente viajero b) Árbol de búsqueda ilustrando las rutas y los costos que podría seguir el agente viajero c) Solución por la búsqueda Viterbi beam [15].

la ecuación. Además la diferencia en el rango dinámico de las dos cantidades es bastante notoria, en particular cuando se utilizan HMMs continuos. Una manera de balancear las dos probabilidades es añadirle un *peso al modelo del lenguaje*, LW, para incrementar la probabilidad $P(\mathbf{W})$ a la potencia $P(\mathbf{W})^{LW}$. El peso del modelo del lenguaje se determina empíricamente para optimizar el desempeño del reconocimiento en un conjunto de desarrollo. Puesto que las probabilidades del modelo acústico están subestimadas, el peso del modelo del lenguaje (LW) típicamente es mayor que uno y para *Sphinx* oscilan en el rango de 6 a 11 [12].

4.4. Tasas de error

Las métricas de evaluación estándar que generalmente se utilizan para probar el desempeño de un sistema de reconocimiento de voz son dos: la *tasa de error de sentencias* (Sentence Error Rate) y la *tasa de error de palabras* (*Word Error Rate*). La tasa de error de sentencias se define como el porcentaje del número de oraciones mal reconocidas del conjunto de pronunciaciones probadas.
La tasa de error de palabras se basa en qué tanto difiere la secuencia de palabras devuelta por el reconocedor, generalmente llamada *secuencia o cadena hipotética de palabras*, con respecto a otra secuencia correcta o *transcripción de referencia* [20]. Dada tal transcripción correcta, el primer paso para la obtención de la tasa de error es el cálculo de la *distancia de edición mínima* que existe entre la secuencia de palabras hipotética y la secuencia de palabras correctas, es decir, se realiza un proceso de alineación de la secuencia de palabras reconocidas contra la secuencia correcta de palabras. El resultado de este cómputo además del número de palabras correctas, dado como *porcentaje de palabras correctas* (*Percent Correct*), también será el número mínimo de *palabras sustituidas*, *palabras insertadas* y de *palabras borradas* que son necesarias para encontrar las coincidencias entre la secuencia correcta y la secuencia hipotética. Así se generan tres posibles errores: el *error por inserción de palabra*, el *error por sustitución de palabra* y el *error por borrado de palabra*. El error por inserción de palabras se da cuando se añade a la secuencia hipotética de palabras una palabra extra; el error por sustitución se da cuando una palabra incorrecta es colocada en lugar de una correcta y el error por borrado se da cuando una palabra correcta se omite en la oración reconocida.

La probabilidad del modelo del lenguaje también tiene la función de penalizar cuando se inserta una nueva palabra o palabras ya existentes [15]. En particular, cuando se usa un modelo de lenguaje con distribución uniforme, la probabilidad del modelo del lenguaje puede ser vista como una pena por inserción de una nueva palabra. Si esta penalización es grande, el decodificador preferirá pocas palabras que tengan mucha longitud y si esta penalización es pequeña, el decodificador permitirá un gran número de palabras cuya longitud sea corta. Al variar el peso del modelo del lenguaje para compensar la probabilidad subestimada del modelo acústico, se tendrá un efecto colateral por ajustar

la penalización por insertar una nueva palabra. Así la contribución al modelo del lenguaje se convierte en

$$P(\mathbf{W})^{LW}\mathbf{IP}^{N(W)} \tag{4.2}$$

donde IP es la penalización por inserción de una nueva palabra, generalmente $0 < IP \leq 1.0$ y $\mathbf{N}(\mathbf{W})$ es el número de palabras en la oración $\mathbf{W}$. La pena por inserción también se determina de manera empírica para optimizar el desempeño del reconocimiento en un conjunto de desarrollo; en el caso de *Sphinx* se utiliza el valor de 0.7 por omisión.

La manera de determinar la tasa de error de palabras por efecto de ambos tipos de errores se da al considerar la ecuación (4.3) [23].

$$WER = \frac{S \ + \ I \ + \ D}{H} * 100\,\% \tag{4.3}$$

donde WER es la tasa de error de palabras, S es el número total de errores de sustitución, I es el número total de palabras insertadas, D es el número total de palabras omitidas y H es número total de palabras en la oración de referencia. Otra medida ampliamente aceptada en reconocimiento de voz es la *precisión de palabra* (*Word Accuracy*). El cálculo del porcentaje de palabras correctas y la precisión de palabra se determina por medio de las ecuaciones (4.4) y (4.5) donde PC es el porcentaje de palabras correctas, WC es el número de palabras correctas en la oración y AC es la precisión de palabra.

$$PC = \frac{WC}{H} * 100\,\% \tag{4.4}$$

$$AC = (1 - WER) * 100\,\% \tag{4.5}$$

Por otra parte, al proceso de alineación también se le conoce como el *problema de máxima coincidencia de subcadenas*, que puede ser manejado mediante programación dinámica [15]. Sea la secuencia de palabras correctas $w_1, w_2, \ldots, w_n$, donde w_i denota la i-ésima palabra en la cadena de palabras correctas, y sea $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_m$ la cadena de palabras reconocidas, donde $\hat{w}_i$ denota la i-ésima palabra en la cadena de palabras reconocidas. Denotamos a $R[i, j]$ como el error mínimo de alineación de las subcadenas $w_1, w_2, \ldots, w_n$ contra las subcadenas $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_m$. La alineación optima y la tasa de error asociada, $R[n, m]$, para la cadena de palabras correctas $w_1, w_2, \ldots, w_n$ y la cadena de palabras reconocidas se obtiene mediante el algoritmo de programación dinámica, que se describe en el algoritmo 4.1.

Algoritmo 4.1 Algoritmo para medir la tasa de error

tbhp

Paso 1: *Inicialización*

$R[0,0] = 0$ $R[i,j] = \infty$ $si (i < 0)$ o $(j < 0)$ $B[0,0] = 0$

Paso 2: *Iteración*

para $i = 1,\ 2, \ldots,\ n$ **hacer**

para $j = 1,\ 2, \ldots,\ m$ **hacer**

$$R[i,j] = \text{mín} \begin{array}{ll} R[i-1,j]+1 & (borrado) \\ R[i-1,j-1] & (coincidencia) \\ R[i-1,j-1]+1 & (sustitución) \\ R[i,j-1] & (inserción) \end{array}$$

$$B[i,j] = \begin{cases} 1 & (borrado) \\ 2 & (coincidencia) \\ 3 & (sustitución) \\ 4 & (inserción) \end{cases}$$

fin para

fin para

Paso 3: *Recursividad hacia atrás y Terminación*

`tasa de error` $= 100\,\% \times \frac{R[n,m]}{n}$

`ruta óptima hacia atrás` $= (s_1,\ s_2, \ldots,\ 0)$

`donde` $s_1 = B[n,m]$,

$$s_t = \begin{bmatrix} B[i-1,j] & si\ s_{i-1} = 1 \\ B[i,j-1] & si\ s_{i-1} = 2 \\ B[i-1,j-1] & si\ s_{i-1} = 3\ \texttt{o}\ 4 \end{bmatrix} \textbf{ para } t = 2, \ldots \textbf{ hasta } s_t = 0$$

4.5. Generación de hipótesis y de celosias

Como ya se mencionó en el capítulo 1 la manera eficiente de resolver el problema de decodificación es haciendo uso del algoritmo de Viterbi, tratado anteriormente. Sin embargo, en la práctica, una implementación directa del algoritmo de Viterbi se vuelve compleja e inmanejable en reconocimiento de voz continua, donde los modelos de topología, las restricciones del lenguaje y la necesidad de limitar los cálculos deben ser tomados en cuenta. Por fortuna, la mayoría de esta complejidad puede ser eliminada al cambiar un punto de vista [4]:

Primero, la topología HMM puede ser hecha en forma explícita al construir una red de reconocimiento. Para las aplicaciones orientadas a alguna tarea, esta red puede representar las locuciones que el usuario tiene permitido decir, es decir, puede representar una gramática de reconocimiento. Para aplicaciones de gran vocabulario, típicamente consistirá de todas las palabras del vocabulario colocadas en paralelo en un ciclo. En ambos casos, las palabras se representan

como una secuencia de modelos de fonos como se definen en el diccionario de pronunciación, como se observa en la figura 4.3, y cada modelo de fono consiste de una secuencia de estados como se indica por el diagrama de la línea punteada. Si una palabra posee múltiples pronunciaciones, éstas simplemente se colocan en paralelo.

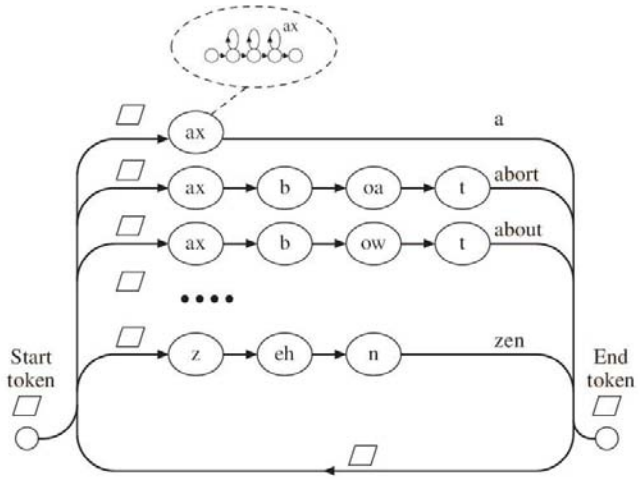


Figura 4.3: Red básica de reconocimiento [4].

Dada esta red, durante la búsqueda en cualquier instante de tiempo t, una hipótesis sencilla consiste de una ruta a través de la red que representa una alineación de los estados con los vectores de características $o_1, \ldots, o_t$, comenzando en el estado inicial y finalizando en el estado j y que tienen una probabilidad logarítmica de $\log \delta_t(j)$. Esta ruta puede volverse explícita por medio de un *token*, que consiste de un par de valores $< \log P,\ liga >$, donde $\log P$ es la probabilidad logarítmica o *puntaje* (*score*) y la liga es un apuntador a un registro del historial de información. Cada nodo de la red corresponde a un estado HMM y puede almacenar un sólo token. Así, el reconocimiento se lleva a cabo al propagar estos tokens alrededor de la red.
El algoritmo de Viterbi ahora puede aplicarse al reconocimiento de voz continua como el algoritmo *token passing*, mostrado en el algoritmo 4.2. El término *nodo* se refiere al nodo de una red que corresponde, a su vez, a un sólo estado HMM. Estos nodos corresponden, ya sea , a un estado de entrada, a un estado emisor o a un estado de salida. En esencia, los tokens son pasados de un nodo a otro y en cada transición el puntaje del token es actualizado.

Cuando un token pasa desde la salida de una palabra al comienzo de la siguiente, su puntaje es actualizado por la probabilidad del modelo del lenguaje más cualquier penalización por inserción de palabra. Al mismo tiempo la transición es guardada en un registro R que contiene una copia del token, el instante de

Algoritmo 4.2 Algoritmo token passing básico

Paso 1: *Inicialización*
Poner un token de inicio $< \log(1),\ 0 >$ en el nodo de entrada de la red;
Poner tokens nulos $< \log(0),\ 0 >$ en todos los otros nodos;
Paso 2: *Iteración*
para cada instante $t = 1,\ 2, \ldots,\ T$ **hacer**
  Paso 2.1**:** *Propagación de los tokens de las palabras internas*
  para cada nodo j que no es de entrada **hacer**
    $maxP = \log(0)$
    para cada nodo predecesor i **hacer**
      Token temporal $Q\ =\ Q_i$
      $Q.logP\ +=\ \log(a_{ij})$ $[+\log(b_j(O_t))$ *si* j *es emisor*$]$
      si $Q.logP > maxP$ **entonces**
        $Q_j = Q;\ maxP = Q.logP;$
      fin si
    fin para
  fin para
  Paso 2.2**:** *Copia los tokens de las salidas de la palabra interna a las entradas siguientes*
  Paso 2.3**:** *Token de palabra externa a partir de la propagación*
  para cada palabra w con nodo de entrada j **hacer**
    Paso 2.3.1
    $maxP = \log(0);$
    Paso 2.3.2
    para cada palabra precedente u con nodo de salida i **hacer**
      Token temporal $Q\ =\ Q_i$
      $Q.logP\ +=\ \alpha \log p(w|u) + \beta$
      si $Q.logP > maxP$ **entonces**
        $Q_j = Q;\ maxP = Q.logP;\ u' = u;$
      fin si
    fin para
    Paso 2.3.3**:** *Registra la decisión del límite de palabra*
    Crea un registro R;
    $R.Q = Q_j;\ R.t = t;\ R.word = u'$
    $Q.link =\uparrow R;$
  fin para
  Paso 2.4**:** *Poner un token nulo en el nodo de entrada de la red*
fin para
Paso 3 *Terminación*
El token en el estado de salida de la red en el instante T representa la mejor ruta

tiempo actual y la identidad de la palabra precedente. El campo de *liga* se actualiza para apuntar al registro R. Como cada token viaja a través de la red, acumula una cadena de estos registros. El mejor token al instante T que esté en un nodo de salida válido puede ser examinado y rastreado hacia atrás para recuperar la secuencia de palabras más probable y sus límites de tiempo.
El algoritmo token passing y la red de reconocimiento asociada son una implantación exacta del principio de programación dinámica visto en el capítulo 1. Para convertir lo anterior en un decodificador práctico para reconocimiento de voz, se requiere de lo siguiente:

1. Para eficiencia computacional, sólo los tokens que tienen alguna probabilidad de estar en la mejor ruta deberían ser propagados. Así, en cada ciclo de propagación, la probabilidad logarítmica del token más probable es registrada. Todos aquellos tokens que caigan por debajo de una constante serán eliminados. Este resultado es denominado como la *busqueda por haz beam search* y la constante es denominada el *ancho del haz* (*beamwidth*).

2. Como consecuencia de la búsqueda por haz el 90 % del cálculo se lleva para obtener los primeros dos fonos de cada palabra, después del cuál, la mayoría de los tokens caen fuera del haz y son podados. Para explotar esto, la red de reconocimiento debe de tener una estructura de árbol, de tal manera que los fonos iniciales estén compartidos, como se observa en la figura 4.4

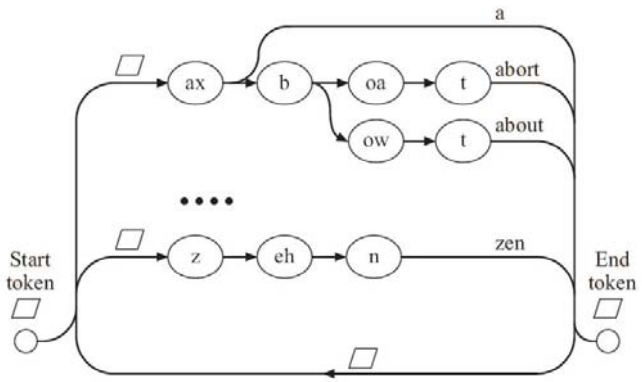


Figura 4.4: Red de reconocimiento con estructura de árbol [4].

3. Sin embargo, compartir fonos iniciales hace imposible aplicar una probabilidad exacta del modelo del lenguaje durante la propagación de tokens en palabras externas, puesto que la identidad de la palabra siguiente se desconoce. El simple retraso de la aplicación del modelo del lenguaje hasta que se haya alcanzado el fin de la siguiente palabra no es una opción, debido a que podría dejar sin efecto al podado. En lugar de ello, se debe

adoptar una aproximación incremental, en la que se tome la probabilidad del modelo del lenguaje como la probabilidad máxima posible dado el conjunto de las siguientes palabras posibles. Como los tokens se mueven a través de un grafo de palabras con estructura de árbol, el conjunto de las posibles palabras siguientes se reduce en la transición de cada fono y la probabilidad del modelo del lenguaje puede ser actualizada con un estimado más preciso.

4. Para obtener un mayor desempeño, los HMMs ligados en la red de reconocimiento deben ser dependientes del contexto, esta dependencia debe extender los límites de las palabras. A simple vista, ésto requiere de una expansión masiva en la red, pero, de hecho, es posible una representación estática y compacta de la red por medio de trifonemas de palabras cruzadas, como se vió en el capítulo 2.

5. El principio de programación dinámica se basa en el hecho de que la ruta óptima en cualquier nodo puede ser extendida si se sabe solamente la información del estado dado en ese nodo. El uso de modelos de lenguaje basado en $N-$gramas presenta un problema aquí, puesto que pueden ser necesarios nodos de red únicos para distinguir todas los posibles $N-1$ historiales de palabras, lo que para decodificadores de gran vocabulario no es tratable. Así, el algoritmo 4.2 sólo trabaja con modelos de lenguaje basado en bigramas. La manera sencilla de resolver ésto es almacenando múltiples tokens en cada estado, permitiendo rutas en con diferentes historiales que sean tratadas en paralelo. La propagación de tokens require, entonces, de operaciones de mezcla y ordenamiento que, aunque son costosas computacionalmente hablando, son manejables.

La descripción anterior de un decodificador de gran vocabulario cubre todos los elementos esenciales necesarios para realizar reconocimiento de voz continua en tiempo real utilizando una sóla pasada sobre los datos. Para la transcripción de la voz en modo batch se pueden hacer mejoras significativas en la precisión al llevar a cabo múltiples pasadas sobre los datos. Para realizarlo, el decodificador debe ser capaz de generar y guardar múltiples hipótesis de reconocimiento. Una estructura compacta y eficiente de hacer esto es mediante una *celosía de palabras* (*word lattice*). Una celosía de palabras consiste de un conjunto de nodos que representan instantes de tiempo y un conjunto de arcos de expansión que representan hipótesis de palabras. Un ejemplo se muestra en la figura 4.5a. Además de los identificadores (ID's) de las palabras, mostradas en la figura, cada arco también puede llevar información acerca del puntaje, tales como los puntajes de los modelos acústico y del lenguaje. Las celosías se forman por medio del mecanismo de "registro de la decisión del límite de palabra", señalado en el algoritmo 4.2, excepto que se registran todos los tokens de fin de palabra, en lugar de registrar solamente el mejor token que está siendo propagado a los siguientes nodos de entrada de las palabras. La calidad de las celosías generadas en el esquema simple de Viterbi de un solo token será muy pobre porque muchas de las segundas mejores rutas, cercanas a la palabra en sí, habrán sido podadas

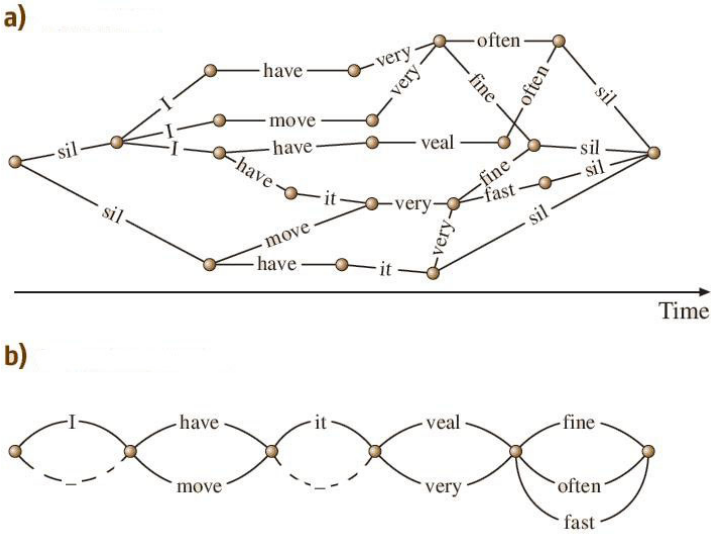


Figura 4.5: Ejemplos de a) red de palabras y de b) red de confusión [4].

al aplicar el principio de programación dinámica. El decodificador de múltiples tokens no padece de este problema, especialmente si se utilza con un modelo del lenguaje basado en bigramas de corta extensión.
Las celosías son extremadamente flexibles. Por ejemplo, pueden obtenerse nuevos puntajes al ser usadas como entrada a la red de reconocimiento y pueden ser expandidas para permitir nuevo puntaje por un modelo del lenguaje de mayor orden. Las celosías también pueden ser simplificadas en una representación muy eficiente denominada *red de confusión* (*confusion network*), como se ilustra en la figura 4.5b, donde los arcos con líneas punteadas indican transiciones nulas. En un red de confusión los nodos ya no corresponden a instantes discretos de tiempo, en lugar de ello simplemente cumplen con las restricciones de la secuencia de palabras. Así, los arcos paralelos en la red de confusión no necesariamente corresponden al mismo segmento acústico. Sin embargo, la mayoría del tiempo los traslapes son suficientes para habilitar arcos en paralelo que sean vistos como hipótesis que compiten en la red. Una red de confusión tiene la propiedad de que, para cada ruta que cruza a través de la celosía original, existe su ruta correspondiente que cruza a través de la red de confusión. Cada arco en la red de confusión lleva la probabilidad a posteriori de su correspondiente palabra w. Esta probabilidad se calcula al hallar la *probablidad de la liga* (*link probability*) de la palabra w en la celosía haciendo uso de un procedimiento hacia adelante-hacia atrás, sumando sobre todas las ocurrencias de w y luego normalizando de tal manera que los arcos de las palabras que compiten en la red de confusión sumen 1. Las redes de confusión pueden ser utilizadas para decodificación de errores de palabras mínimos. Para proporcionar puntajes efi-

cientes y para mezclar las salidas de diferentes decodificadores.
Finalmente, debe notarse que lo anterior hace referencia a una aproximación específica de decodificación. Si la decodificación de Viterbi fuera el único requerimiento, entonces habría pequeñas variaciones entre las implementaciones de los decodificadores. Sin embargo, el requerimiento para dar soporte a cruce de palabras de modelos acústicos dependientes del contexto y a modelos de lenguaje de gran expansión han conducido a una variedad de estrategias de diseño. Por ejemplo, en vez de tener múltiples tokens, la red de estados puede ser expandida dinámicamente para representar de manera explícita tanto al cruce de palabras acústico hipotético y a los contextos del modelo del lenguaje de gran expansión. Estos decodificadores dinámicos son más flexibles que las redes de decodificación estática, pero son más difíciles de implementar. Los avances recientes en la tecnología de los transductores de estados finitos pesados ofrece la posibilidad de integrar los modelos acústicos, los modelos de pronunciación, las probabilidades del lenguaje, etc. en una red simple muy grande pero altamente optimizada. Esta aproximación ofrece tanto flexibilidad y eficiencia y por lo tanto es extremadamente útil en investigación y en aplicaciones prácticas. Sin embargo, debido a los alcances de este trabajo no serán tratadas aquí.

5

El sistema de reconocimiento de voz: *Sphinx*

En este capítulo se mostrarán los aspectos fundamentales del sistema *Sphinx*. Se hará mención de los pasos necesarios para el entrenamiento y para el reconocimiento. Asimismo se tratará acerca del conjunto de herramientas *CMU SLM* necesarias para realizar el modelado del lenguaje. Cabe mencionar que para la realización de este capítulo me apoyé tanto en la documentación de Sphinx [42, 43, 44, 45, 47], como en la labor que inició el M.I. Omar Nieto Crisóstomo y cuyos apuntes [29] me dieron la pauta para llevar a cabo el proceso de entrenamiento del sistema *Sphinx*.

5.1. Visión general del entrenamiento

El proceso de entrenamiento en el sistema *Sphinx* se resume en las figuras 5.1 y 5.2 y se tratará con más detalle en las secciones siguientes. Se utilizarán como ejemplos aquellos comandos correspondientes al entrenamiento que se hizo de las 4 frases en español que se desean reconocer. Cabe mencionar que para el entrenamiento de las 109 palabras del español se aplicarán comandos similares en cada etapa.

5.1.1. Creación del archivo de definición del modelo CI

El primer paso consiste en preparar un *archivo de definición del modelo* para los fonemas independientes del contexto (CI). La función de este archivo es la de definir o proporcionar un identificador numérico único para cada estado de cada HMM que se vaya a entrenar, además, proporcionarle un orden, que es el que se seguirá en la escritura de los parámetros del modelo en sus respectivos archivos. Durante el entrenamiento los estados solamente serán referenciados por estos identificadores, así, el archivo de definición del modelo especifica una parte de la arquitectura de los HMM.

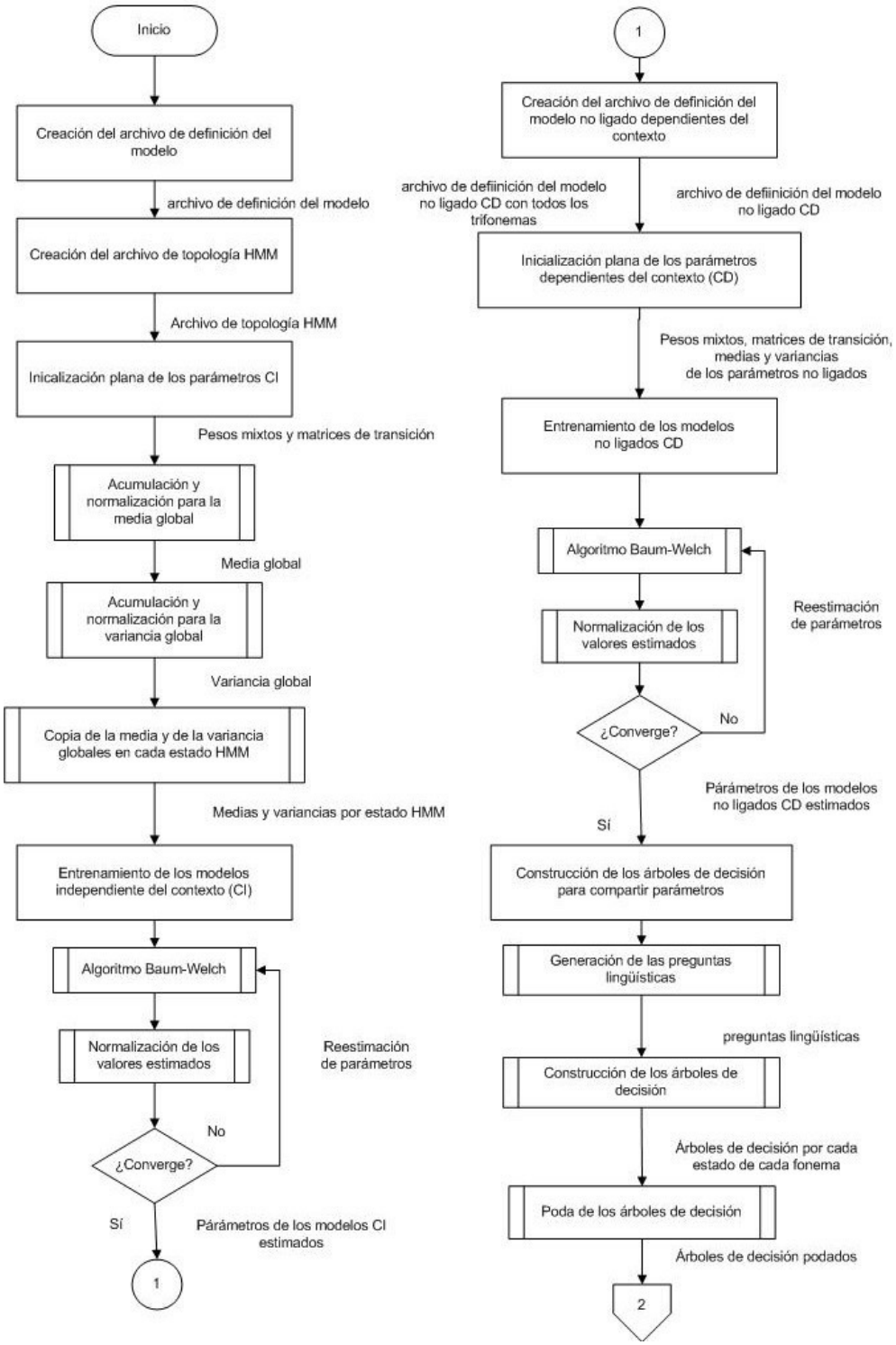


Figura 5.1: Entrenamiento de HMM continuos en *Sphinx*.

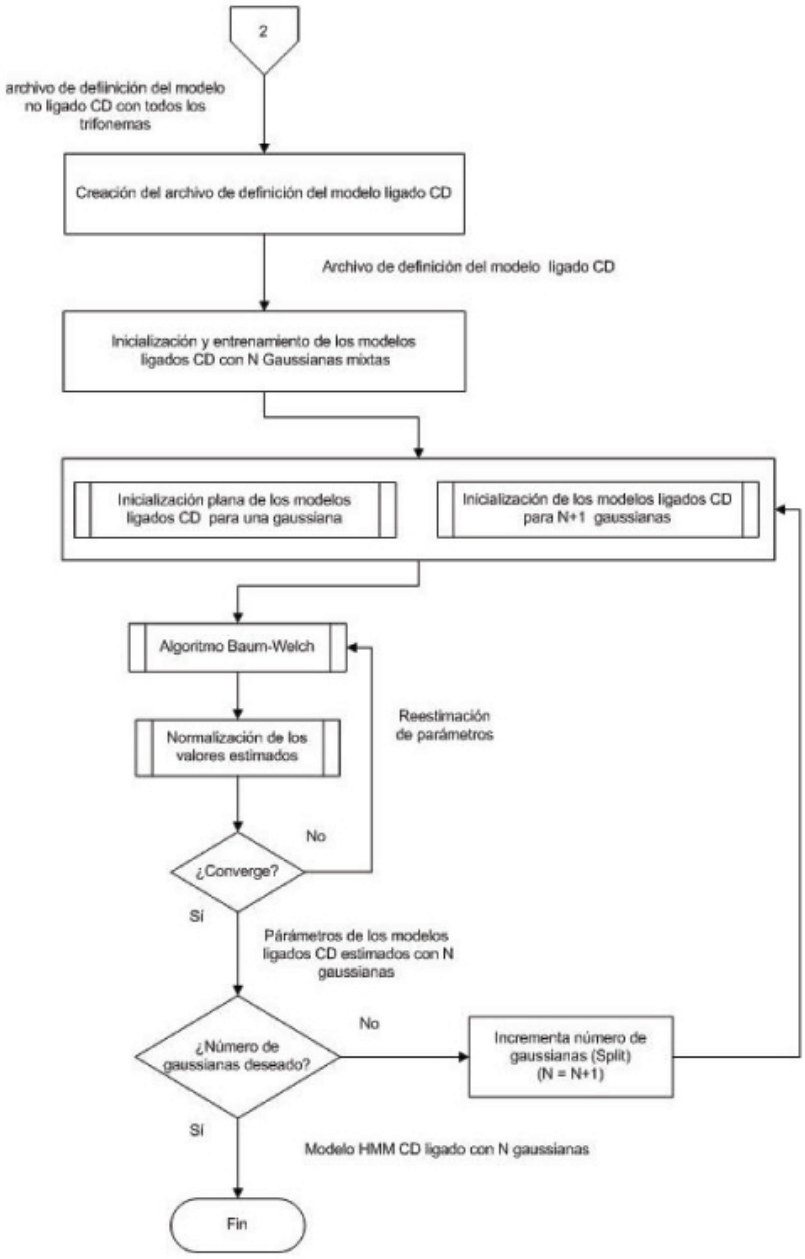


Figura 5.2: Entrenamiento de HMM continuos en *Sphinx* (cont.)

Para generar el archivo de definición del modelo CI, se hará uso de la aplicación `mk_mdef_gen`, cuyos parámetros se describen en la tabla E.5 del apéndice E.

5.1.2. Creación del archivo de topología HMM

El archivo de topología HMM es un archivo único que contiene a la matriz de definición de la topología y es común para todos los HMM. Contiene una matriz con entradas booleanas. Cada entrada de esa matriz indica si en el HMM está permitido o no realizar una transición de un estado renglón i, señalado por el número del renglón, a un estado columna j, señalado por el número de la columna. Por ejemplo, un HMM de 3 estados que no tenga permitido ninguna transición entre estados que no sean adyacentes o que no sean el mismo estado podría tener un archivo de topología con la siguiente información:

```
#
# 3-state Bakis topology HMM with non-emitting last state
# These values are normalized so that rows sum to one.
#
# NO COMMENTS BETWEEN # OF STATES AND TRANSITION MATRIX
#
#
#Version number
0.1
# Number of states per model followed by transition matrix
4
1.0 1.0 0.0 0.0
0.0 1.0 1.0 0.0
0.0 0.0 1.0 1.0
# Last state has no outgoing arcs unless
# embedded in a sentence hmm structure
```

Observe que las líneas que empiezan con # se consideran como comentarios. El número 0.1 indica la versión del script que se usó para generar dicho archivo. El número 4 corresponde al número total de estados en el HMM. Cabe señalar que *Sphinx* agrega automáticamente un estado adicional considerado como un estado nulo o no-emitido, en este caso corresponde al estado número 4, que se usa como terminación del HMM de 3 estados. El último estado $(3, 4)$ no contiene arco de salida a menos que se encuentre concatenado en una oración con estructura HMM. El valor de 1.0 en cada celda (i, j) de la matriz significa que en ese estado se permite una transición del estado i al estado j. En el caso de la celda $(1, 1) = 1.0$ indica una transición hacia sí mismo. Por consiguiente, la matriz de transición estimada para cualquier fonema podría ser usada para asignar un valor de transición de probabilidad en lugar de estos valores booleanos. Cuando el valor de la celda sea de 0.0, su correspondiente transición de probabilidad no será estimada.

Se puede escribir el archivo de topología en forma manual o mediante el script `maketopology.pl` proporcionado por *Sphinx*. Los argumentos de este script se detallan en la tabla E.1 del apéndice E.

5.1.3. Inicialización plana de los parámetros CI

Para comezar el entrenamiento de los modelos HMM CI, se necesita de cuatro archivos donde se definan a los parámetros del modelo CI. Tales archivos son:

- `mixture_weights`: Los pesos asignados a cada Gaussiana en las correspondientes gaussianas mixtas para cada estado.
- `transition_matrices`: La matriz de probabilidades de transición de estados.
- `means`: Las medias de todas las Gaussianas.
- `variances`: Las variancias de todas las Gaussianas.

Estos archivos requieren de entradas iniciales por lo que se llevan a cabo sendos procesos de inicialización, como se indica en los apartados siguientes:

Inicialización de los pesos mixtos y las matrices de transición

Los archivos `mixture_weights` y `transition_matrices` se inicializan usando la aplicación `mk_flat`. Sus argumentos se discutirán en el apéndice E en la tabla E.11.

A grandes rasgos, la aplicación `mk_flat` calcula la información contenida en la matriz de transición inicial de la siguiente manera: Realiza la lectura del archivo de topología. Después, por cada renglón de la matriz de transición, cuenta los elementos donde existe una transición (celdas con valor de 1.0) y obtiene la suma acumulada en ese renglón. Cuando termina la suma, divide a todos los elementos de ese mismo renglón por dicha suma acumulada. Lo anterior es una forma de garantizar que todas las transiciones posean la misma probabilidad. La figura 5.3 muestra un ejemplo de este procedimiento para la matriz de topología definida en la sección anterior.

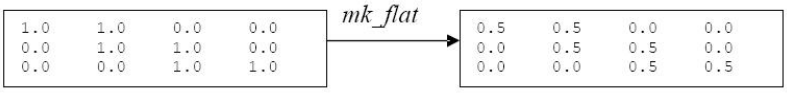


Figura 5.3: Obtención de matriz de transición [29].

Por otra parte, cada peso mixto se calcula mediante el algoritmo 5.1. De acuerdo con este algoritmo los pesos mixtos iniciales se calculan al dividir cada flujo de características de cada estado por la cantidad de densidades que se manejan (1 para el caso continuo).

Inicialización de la media y la variancia globales

Para inicializar la media y la variancia, los valores globales de esos parámetros se estiman primero y después se copian en localidades apropiadas dentro

Algoritmo 5.1 Algoritmo para calcular los pesos mixtos

para $i = 0, \ldots,\ num\_estados - 1$ **hacer**
  para $j = 0, \ldots,\ num\_características - 1$ **hacer**
    para $k = 0, \ldots,\ num\_densidades - 1$ **hacer**
      $mixw[i][j][k] = 1/num\_densidades$
    fin para
  fin para
fin para

de los archivos de parámetros. La media global se calcula al sumar todos los vectores que se tienen en los archivos de características (archivos con extensión `mfc`). Como el valor de la media es, por lo regular, un número muy grande el cálculo se divide en varias partes. Así, se le indica a *Sphinx* la cantidad de partes en que se quiere dividir esta operación. Dependiendo de las capacidades de cómputo que se tengan, *Sphinx* acumula o reúne los vectores para cada parte en forma separada y los escribe en la máquina en un buffer intermedio .

Los archivos `means` y `variances` se inicializan por medio de la aplicación `init_gau`, cuyos argumentos se discutirán en la tabla E.12 del apéndice E.

Acumulación y normalización para la media global

Durante esta primera etapa de inicialización, la aplicación `init_gau` se encarga de calcular la acumulación para la media global de todas las palabras del entrenamiento. En términos generales, el procedimiento lleva a cabo los siguientes pasos:

Para cada línea en el archivo de control (argumento en `ctlfn`), se realiza lo siguiente:

- Se realiza la lectura de los coeficientes cepstral correspondientes a la línea actual.
- Se realiza la compresión del silencio, si es indicado en el parámetro `silcomp`. Este parámetro puede tomar uno de 3 valores [29]:

  current. Se encarga de eliminar las tramas que tienen energía menor que un umbral calculado mediante un porcentaje de la energía normalizada, como se muestra en la figura 5.4. Para eliminar la trama de silencio n se debe cumplir que las tramas vecinas $n-1,\ n-2,\ n+1$ y $n+2$ también deben tener valores de energía menores que el umbral.

  sildelfn. Elimina un número de tramas definidas por dos arreglos, uno para el comienzo de la eliminación y el otro para el fin. Por ejemplo,

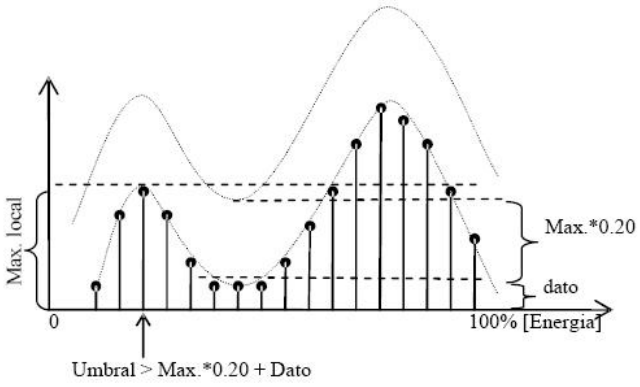


Figura 5.4: Umbrales de `silcomp` opción current [29].

para eliminar el bloque i de tramas se asigna:

$$\begin{aligned} del\_b[i] &= \text{índice de la trama de inicio} \\ del\_e[i] &= \text{índice de la trama final} \end{aligned}$$

Las tramas que se encuentran en este intervalo se eliminarán por completo.

none. No elimina silencios.

- Enseguida se efectúa la normalización de los vectores utilizando los argumentos `cmn` y `varnorm`, que se encargan del control de normalización de la media y el control de normalización de la variancia, respectivamente. El argumento `varnorm` toma los valores "yes" o "no". El argumento `cmn` tiene los valores:

  current Realiza la normalización utilizando la media global de acuerdo con la ecuación (5.1)

$$media[i] = \frac{\sum_{k=0}^{\#tramas-1} mfcc[k][i]}{\#tramas} \quad i = 0, \ldots, \#coefs - 1 \qquad (5.1)$$

  Para el caso en que el argumento `varnorm` tome el valor de "yes", la

normalización se hace de la siguiente forma:

$$var[i] = \sqrt{\frac{\sum_{k=0}^{\#tramas-1} (mfcc[k][i] - media[i])^2}{\#tramas}} \quad i = 0, \ldots, \#coefs - 1 \tag{5.2}$$

$$mfcc_{Norm}[k][i] = \frac{mfcc[k][i] - media[i]}{var[i]} \quad \begin{matrix} k = 0, \ldots, \#tramas - 1 \\ i = 0, \ldots, \#coefs - 1 \end{matrix} \tag{5.3}$$

En cambio, cuando se deja el valor "no", que es el valor de `varnorm` por defecto, la normalización se realiza como sigue:

$$mfcc_{Norm}[k][i] = mfcc[k][i] - media[i] \quad \begin{matrix} k = 0, \ldots, \#tramas - 1 \\ i = 0, \ldots, \#coefs - 1 \end{matrix} \tag{5.4}$$

prior Este argumento indica que la normalización se calculará utilizando la media de la oración anterior. El algoritmo 5.2 muestra esta forma de normalización.

Algoritmo 5.2 Algoritmo para calcular la media con base en la media de la oración anterior

Paso 1: *Inicialización*
$mediaAct_0[i] = 0 \quad i = 0, \ldots, \#coefs - 1$
$sum_0[i] = 0 \quad i = 0, \ldots, \#coefs - 1$
Paso 2: *Iteración*
para $m = 1, \ldots, \ \#sentencias$ **hacer**
  para $k = 0, \ldots, \ \#tramas - 1$ **hacer**
    para $i = 0, \ldots, \ \#coefs - 1$ **hacer**
      $mfcc_{Norm_m}[k][i] = mfcc_m[k][i] - mediaAct_{m-1}[i]$
    fin para
  fin para
  para $i = 0, \ldots, \ \#coefs - 1$ **hacer**

$$sum_m[i] = \frac{\sum_{k=0}^{\#tramas_m - 1} mfcc_m[k][i]}{\#tramas_m}$$

$$mediaAct_m[i] = \frac{mediaAct_{m-1}[i] - sum_m[i]}{2}$$

  fin para
fin para

none Indica que no se realiza normalización

- Se efectúa el control automático de ganancia con el argumento `agc` si éste se encuentra activo. El argumento `agc` toma los siguientes valores:

  noise Calcula el promedio de la energía del ruido de las tramas que tengan una energía menor a una energía mínima más un umbral del

ruido (umbral del ruido = 0.2). Este promedio es substraído a todos los valores de la energía (En los MFCC, los coeficiente $mfcc[k][0]$ determinan el valor de la energía logarítmica), es decir,

$$\begin{aligned} \forall q \text{ donde } mfcc[q][0] &< \acute{Energia}\ m\acute{inima} + umbral\ de\ ruido \\ nivel\ de\ ruido &= \frac{\sum_q mfcc[q][0]}{\#\ total\ de\ elementos\ de\ q} \quad (5.5) \\ mfcc[k][0] &= mfcc[k][0] - nivel\ de\ ruido \quad (5.6) \end{aligned}$$

max El valor máximo de la energía es substraído a cada componente de la energía

$$\begin{aligned} mfcc[k][0] &= mfcc[k][0] - \acute{Energia}\ m\acute{axima} \quad (5.7) \\ & k = 0, \ldots, \#tramas - 1 \quad (5.8) \end{aligned}$$

emax Por medio del algoritmo 5.3 determina el valor que será substraído a cada componente de energía.

Algoritmo 5.3 Algoritmo de valor substraído para la energía

thp

Paso 1: *Inicialización*
$target\_max = 1.0$; $max = 1.0$; $\Delta max = 0.0$;
$decay = 1.0/3000$; $min\_max = -0.5$
Paso 2: *Iteración*
para $k = 0, \ldots,\ \#Tramas$ **hacer**
  si $mfcc[k][0] > target\_max$ **entonces**
    $target\_max = mfcc[k][0]$
    si $\Delta max < \frac{target\_max - max}{100}$ **entonces**
      $\Delta max = \frac{target\_max - max}{100}$
    fin si
  fin si
  si $target\_max \geq max$ **entonces**
    $max = max + \Delta max$
  sino
    si $target\_max > min\_max$ **entonces**
      $target\_max = target\_max - decay$
    fin si
    si $max > min\_max$ **entonces**
      $max = max - decay$
      $mfcc[k][0] = mfcc[k][0] - max$;
    fin si
  fin si
fin para

none No se aplica el control automático de ganancia.

- Se calculan los vectores de características a partir de los vectores MFCC, de la siguiente forma: El vector de características final de la voz se crea típicamente aumentando el vector cepstral con la primera y segunda derivadas con respecto al tiempo y después de calcular `cmn` y `agc`, si es que fueron habilitadas esas opciones. El vector característico que se calcula en cada trama tiene dimensión 39, como se ve en la figura 5.5.

  Se observa entonces que los vectores delta son calculados como la diferencia entre los vectores cepstral de dos tramas removidas a cada lado de la trama `f` actual, seguidas de la diferencia entre los vectores delta cepstral de una trama removida a cada lado de la trama `f` actual. El cálculo de los coeficientes de potencia se lleva a cabo aplicando un procedimiento similar puesto que corresponden al primer coeficiente de cada vector de características.

- Después de obtener los vectores de características se realiza una acumulación de cada uno de sus elementos. La figura 5.6 muestra un esquema de cómo se realiza la acumulación para el flujo de datos usado en modelos continuos en *Sphinx* ($1s\_c\_d\_dd$).

Cabe destacar que los valores obtenidos no son propiamente la media, sino la sumatoria de los elementos correspondientes. En el mismo proceso de acumulación, se utiliza un contador que guarda el número de elementos acumulados y que se almacena junto con los valores acumulados. *Sphinx* utiliza un archivo temporal denominado `gauden_counts` donde almacena los valores mencionados.

Después de haber guardado los buffers, el contenido de éstos debe ser *normalizado* o usado para calcular el valor de la media global de los vectores de características. Esto se lleva a cabo a través del ejecutable `norm` con los parámetros que se detallan en la tabla E.14 del apéndice E. `norm` realiza la lectura del buffer intermedio que se encuentra en el archivo `gauden_counts`; después verifica si su contenido ha sido almacenado correctamente y efectúa la normalización de la siguiente manera: Cada elemento de la matriz donde se almacenaron las sumas de los vectores MFCC se dividido por el número de vectores que se utilizaron en la acumulación. El algorimto 5.4 muestra este proceso.

Acumulación y normalización para la variancia global

El siguiente paso consiste en acumular los vectores para el cálculo de la variancia global de todos los vectores de entrenamiento. Esto se logra utilizando por segunda vez la aplicación `init_gau`. Los argumentos de `init_gau` se presentan en la tabla E.12 del apéndice E. En términos generales, `init_gau` ejecuta el siguiente procedimiento:

- Realiza la lectura de la media global, que se encuentra almacenada en el archivo dado por el argumento `-meanfn`.

- Para cada línea en el archivo de control (`-ctlfn`), se realiza lo siguiente:

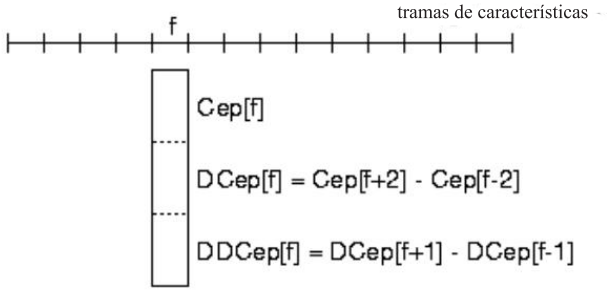


Figura 5.5: Cálculo del vector de características [38].

Algoritmo 5.4 Algoritmo para la normalización de la media

para $i = 0, \ldots,\ num\_estados - 1$ **hacer**
  para $j = 0, \ldots,\ num\_características - 1$ **hacer**
    para $k = 0, \ldots,\ num\_densidades - 1$ **hacer**
      si $dnom[i][j][k] \neq 0$ **entonces**
        para $q = 0, \ldots, veclen[j] - 1$ **hacer**
          $mean[i][j][k][q] = mean[i][j][k][q]/dnom[i][j][k];$
        fin para
      fin si
    fin para
  fin para
fin para

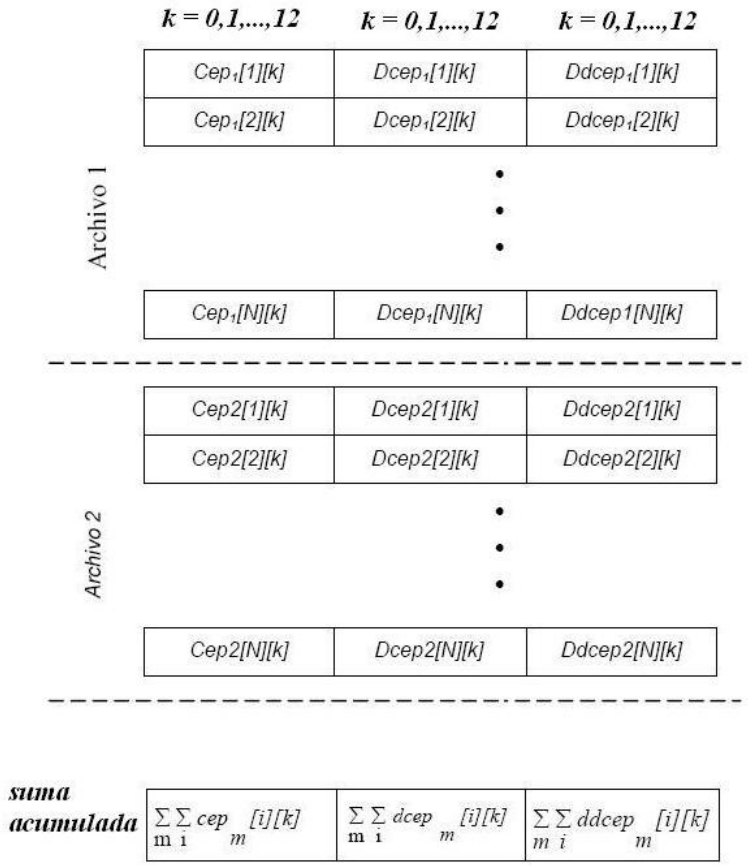


Figura 5.6: Acumulación para la media.

- Se leen los coeficientes Mel-cepstral correspondientes a la línea actual
- Se realiza la compresión de silencio, de acuerdo a como se describió en la sección anterior el argumento `-silcomp`.
- A continuación se efectúa la normalización de los vectores de acuerdo a como estén establecidos los valores de los los argumentos `-cmn` y `-varnorm`, descritos en la sección anterior.
- Se efectúa el control automático de ganancia conforme al valor establecido del argumento `-agc`, tal y como se describió en la sección anterior.
- Se calculan los vectores de características utilizando los vectores MFCC como ya se describió en la sección anterior.
- Despúes, toma el valor de la media global y recolecta cada uno de

los cuadrados de las diferencias entre cada vector y la media global. El algoritmo 5.5 describe la forma de cómo se realiza la obtención de estas diferencias.

- Realiza la sumatoria de las diferencias de los elementos de la variancia. El cálculo de esta acumulación también se puede dar en una o varias partes y se escribe en el buffer respectivo.
- Almacena un contador con el número de elementos acumulados, ya que solamente se realizó la sumatoria.

Algoritmo 5.5 Algoritmo para acumulación de la variancia

para $m = 0, \ldots,\ num\_archivos - 1$ **hacer**
  para $i = 0, \ldots,\ num\_tramas\_del\_archivo\_m - 1$ **hacer**
    para $j = 0, \ldots,\ num\_flujos\_de\_caracter\acute{\imath}sticas - 1$ **hacer**
      {Para la variancia global el num. de densidades = 1}
      para $q = 0, \ldots, veclen[j] - 1$ **hacer**
        $var[j][0][q] = (mfcc[m][i][j][q] - m[j][0][q])^2$
      fin para
    fin para
  fin para
fin para

Nuevamente, una vez que los buffers se han escrito, el contenido de éstos son *normalizados* o usados para calcular el valor de la variancia global. De nueva cuenta se hace uso del ejecutable `norm`, con los argumentos que se detallan en la tabla E.14 del apéndice E, pero haciendo referencia al cálculo de la variancia.

En forma similar a como se hizo en la normalización de la media global, `norm` lee el archivo intermedio `gauden_counts`, verifica si su contenido ha sido almacenado correctamente. Enseguida lleva a cabo la normalización de la siguiente manera: Cada elemento de la matriz donde se almacenaron las sumas para la variancia, se divide por el número de vectores que se utilizaron en la acumulación. Este procedimiento se muestra en el algoritmo 5.6

Copia de la media y variancia globales a cada estado HMM

Después de que la media y la variancia globales hayan sido calculadas, éstas deben ser copiadas en cada estado de los HMM para asignarles, así, sus respectivas medias y variancias. La media global es escrita en una localidad apropiada del estado dentro de un *archivo de medias*. De forma similar la variancia global también se escribe en una localidad apropiada del estado dentro de un *archivo de variancias*. Los archivos planos de las medias y las variancias pueden ser creados usando el ejecutable `cp_parm`. Para poder hacer uso de este software primero es necesario crear un *archivo de mapa de operaciones de copia* (*copy-operations map file*). Dicho archivo se forma de dos columnas: Cada columna se

Algoritmo 5.6 Algoritmo para la normalización de la variancia

para $i = 0, \ldots,\ num\_estados - 1$ **hacer**
  para $j = 0, \ldots,\ num\_características - 1$ **hacer**
    para $k = 0, \ldots,\ num\_densidades - 1$ **hacer**
      si $dnom[i][j][k] \neq 0$ **entonces**
        para $q = 0, \ldots, veclen[j] - 1$ **hacer**
          $var[i][j][k][q] = var[i][j][k][q]/dnom[i][j][k];$
        fin para
      fin si
    fin para
  fin para
fin para

forma por un identificador único o ID de un estado. En la columna de la izquierda se encuentra los $ID's$ de los estados destino, es decir, a donde se copiará el valor global deseado; y en la columna de la derecha se encuentra los $ID's$ de los estados de origen, es decir, aquellos estados que tienen los valores que se desean copiar. Por ejemplo, un segmento del archivo de mapa de operaciones que es usado en el entrenamiento de las 4 frases del español es el siguiente:

```
0 0
1 0
2 0
3 0
.
.
.
63 0
64 0
65 0
```

En el caso de 22 fonemas independientes del contexto, cada fonema se representa por 3 estados emitidos y uno no emitido. El estado no emitido no tiene parámetros por lo que el archivo consiste de $22 \times 3 = 66$ estados, comenzando en el estado 0 y finalizando en el estado 65, en la columna de estados destino. En la columna de estados de origen solamente se hace referencia al primer estado puesto que hasta el momento solamente se cuenta con un valor global calculado; uno para la media y uno para la variancia.
`cp_parm` requiere de los argumentos descritos en la tabla E.3 del apéndice E. La aplicación tiene que ser ejecutada dos veces, una para copiar la media y la otra para copiar la variancia. A grandes rasgos, dicho software realiza los siguientes procedimientos:

- Realiza la lectura del archivo indicado por el argumento `-gaufn`, que es el archivo donde se almacena el arreglo con la media o con variancia globales, según corresponda. En el caso de la media, el flujo de características se puede observar en la figura 5.5.

- Con el parámetro `-ncbout` se crea un arreglo cuya dimensión corresponde al total de estados emitidos que fueron generados. Cada elemento de este arreglo tiene la estructura de la media o de la variancia globales, según corresponda. Por ejemplo, el caso de la media global se muestra en la figura 5.7.

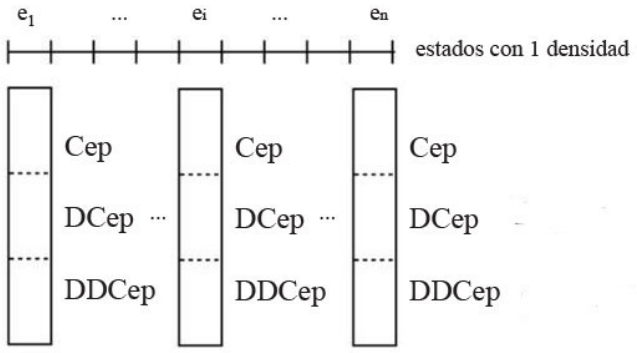


Figura 5.7: Copia de la media global a cada estado HMM.

- Realiza la lectura del archivo del mapa de la operación de copia , dado en el parámetro `-cpopsfn`. A cada estado destino se copia la media o la variancia globales, segun corresponda.
- Finalmente se almacena el arreglo, con las medias o las variancias inicializadas, según corresponda, en el archivo indicado por el argumento `-ogaufn`.

5.1.4. Entrenamiento de los modelos independientes del contexto (CI)

Después de que se ha llevado a cabo la etapa de inicialización plana, se prosigue con el entrenamiento de los modelos acústicos para los fonemas base ó fonemas independientes del Contexto (CI). Este paso es llamado *entrenamiento-CI*, donde los modelos inicializados en forma plana son reestimados por medio del algoritmo Baum-Welch.

Algoritmo Baum-Welch

La reestimación por medio del algoritmo Baum-Welch es un proceso que se realiza en forma iterativa, de manera tal, que dicho algoritmo se tiene que ejecutar en muchos pasos sobre sus datos de entrenamiento. A medida que se lleva a cabo cada uno de estos pasos, se obtiene un conjunto de modelos CI que se ajustan ligeramente más a los modelos de la iteración anterior. Sin embargo, puesto que la función objetivo a maximizar en cada uno de estos pasos es

la probabilidad (likelihood), bastantes iteraciones podrían conducir a modelos que se ajusten demasiado a los datos de entrenamiento, lo cual no es deseable por diversas razones. En el entrenamiento CI típicamente se requieren de 5 a 8 iteraciones del algoritmo Baum-Welch para obtener buenas estimaciones de los modelos CI. Es posible determinar en forma automática el número de iteraciones necesarias [29]; ésto se logra al observar la probabilidad total de los datos de entrenamiento que se ha obtenido al final de la primera iteración, y, después, tomando una decisión con base en una *razón* o *criterio de convergencia* de probabilidades. Este criterio se da simplemente como la proporción de la probabilidad total en la iteración actual con respecto a la probabilidad total obtenida en la iteración previa, por ejemplo, sea P_{n-1} la probabilidad logarítmica por trama de las pronunciaciones de la ejecución anterior, $n-1$, y sea P_n la probabilidad logarítmica por trama con respecto a la ejecución actual, n; entonces la razón de convergencia está dada por

$$\begin{matrix} razón\ de \\ convergencia \end{matrix} = \left(\frac{P_n - P_{n-1}}{abs(P_{n-1})} \right) \tag{5.9}$$

donde cada P_i se define como

$$P_i = \frac{P_{total\ por\ iteración}}{\#\ de\ tramas} \tag{5.10}$$

A medida que se obtienen modelos que se ajustan cada vez más a los datos de entrenamiento, en cada iteración, las probabilidades de los datos de entrenamiento típicamente se incrementan de forma monótona. Por lo tanto, la razón de convergencia se trata de un pequeño número positivo. Conforme avanzan las iteraciones, la razón de convergencia se vuelve cada vez más pequeña, puesto que, cada vez que se obtienen los modelos actuales, la diferencia con respecto a los modelos anteriores también se vuelve cada vez más pequeña. Las razones de convergencia son específicas para cada tipo de datos y para cada tarea, sin embargo, los valores típicos que se pueden utilizar como criterio para detener las iteraciones del algoritmo Baum-Welch en el caso del entrenamiento-CI, oscilan en el rango de 0.001 a 0.1. En el caso de que los modelos estén normalizados con respecto a la variancia, las razones de convergencia son mucho más pequeñas.
La aplicación que se usa para correr una iteración del algoritmo Baum-Welch es llamada `bw`y sus argumentos se describen en las tablas E.15 y E.16 del apéndice E.
A grandes rasgos el software `bw` de *Sphinx* realiza las siguientes etapas de manera general:

Inicialización de estructuras y variables En esta etapa se encarga entre otras cosas de: Realizar la lectura del archivo de definición del modelo e inicializar varias variables con toda la información contenida en este archivo; lee los archivos de pesos mixtos y el archivo de las matrices de transición para obtener sus respectivos valores inicializados; lee también los archivos de las medias y las variancias para obtener sus respectivos valores inicializados; también para cada densidad, calcula el factor de normalización y

la matriz de precisión (inversa de la matriz diagonal de covariancias) para hacer más eficientes los cálculos. Por último, lee los archivo del diccionario de palabras y del diccionario de relleno para alinearlos con sus respectivos modelos.

Estimación de parámetros por máxima probabilidad (MLE) En esta etapa realiza la reestimación con base a todo el cuerpo de entrenamiento, que se encuentra almacenado en distintos archivos, mencionados al inicio de este capítulo: archivos de características, archivo de transcripción y archivo de control.
Para cada línea en el archivo de control lleva a cabo las siguientes tareas:

- Realiza la lectura del archivo respectivo que tiene los coeficientes Mel-Cepstral y en el archivo de transcripción lee la oración correpondiente a esta pronunciación.
- Aplica el control de normalización de la media y el control automático de ganancia, si acaso se han establecido de antemano. Después calcula y almacena el flujo de los vectores de características, como ya se ha señalado anteriormente.
- Construye el modelo acústico de cada pronunciación, creando una lista de todos los fonemas que conforman la pronunciación actual. Este procedimiento se hace de acuerdo a la sección 2.4 del capítulo 2. Además, se almacenan una lista de los estados previos y próximos a cada estado HMM (los estados que presentan una transición a este estado y que parten de este estado), también almacena las respectivas probabilidades de las transición.

Procedimiento hacia adelante Este procedimiento, descrito en el capítulo 1, calcula la variable hacia adelante (forward variable) $\alpha_t(i)$ y los valores de escalamiento para cada instante de tiempo t.

Procedimiento hacia atrás Este procedimiento, descrito en el capítulo 1, calcula la variable hacia atrás $\beta_t(i)$. Al mismo tiempo que se calcula la variable de retroceso, se obtienen los valores estimados de la media, la variancia, los pesos mixtos, etc.

El proceso de estimación de los parámetros de todas las pronunciaciones se puede llevar a cabo en una o varias partes. Cuando se concluye dicho proceso de estimación solamente se obtienen los valores acumulados de los parámetros del modelo HMM y se almacenan en respectivos buffers intermedios a manera de archivos: `mixw_counts`, archivo donde se almacenan la acumulación de los pesos mixtos, `tmat_counts`, archivo para guardar los valores de las matrices de transición, `gauden_counts`, archivo para guardar los valores de acumulación de la media y variancia. Además, también se almacena el valor acumulado del denominador.

Normalización de los valores estimados

Una vez que la aplicación `bw` ha concluido, el siguiente paso para obtener los parámetros estimados es la *normalización*. Los parámetros del modelo final, es decir, las medias, las variancias, los pesos mixtos y las matrices de transición, son calculados con la información contenida en los buffers antes mencionados. De nueva cuenta se recurre a la aplicación `norm` para realizar la normalización.

Recuerde que se requieren de varias iteraciones con las aplicaciones `bw` y `norm` para obtener los modelos independientes del contexto, El criterio para detener las iteraciones depende de si se haya alcanzado o no una razón de convergencia en la probabilidad de los datos de entrenamiento.

Los parámetros del modelo calculados por `norm` en la iteración final, ahora se utilizan para la inicialización de los modelos de fonemas dependientes del contexto (trifonemas) con estados no ligados (untied states). Éste es otro de los pasos más importantes que se encuentran en el proceso de entrenamiento de HMM continuos. Al proceso de entrenamiento de HMMs de trifonemas con estados no ligados se le denominará como *entrenamiento no ligado dependiente del contexto* o *entrenamiento no ligado CD* (*CD untied training*).

5.1.5. Creación del archivo de definición del modelo no ligado dependiente del contexto

En el entrenamiento no ligado dependiente del contexto usualmente se utilizan trifonemas, los cuales son entrenados para todos los fonemas dependientes del contexto que se observen en el cuerpo de entrenamiento.

Para el entrenamiento no ligado CD, se necesita generar un *archivo de definición del modelo para todos los trifonemas que ocurren en el cuerpo de entrenamiento*. Esto se puede realizan de dos formas: una forma por pasos y otra directa. La última es la forma mas eficaz, razón por la que en este trabajo se usa dicha forma. La forma directa se lleva a cabo por medio de la aplicación `mk_mdef_gen`, cuyos parámetros se definen en la tabla E.5 del apéndice E. A grandes rasgos esta aplicación realiza los siguientes pasos:

- Genera una lista de todos los trifonemas posibles en el vocabulario a partir del diccionario. La lista de los fonemas se encuentra en el siguiente formato:

```
fonema1 0 0 0 0
fonema2 0 0 0 0
fonema3 0 0 0 0
fonema4 0 0 0 0
...
```

  Para construir esta lista utiliza la lista de los fonemas del entrenamiento independiente del contexto (CI), respetando el mismo orden en el cual éstos fueron listados.

- Genera un diccionario temporal, que tiene todas las palabras excepto las palabras de relleno. Añade la entrada `SIL` a este diccionario temporal y lo ordena alfabéticamente como sigue:

```
A - - -
A_1 - - -
B - - -
...
P - - -
R - - -
S - - -
SIL - - -
T - - -
U - - -
X - - -
```

- Genera una lista de todos los posibles trifonemas a partir del diccionario temporal. La lista tiene el siguiente formato:

```
A CH E e
A D D i
A I L i
A I S i
A L I e
A L S e
A L U b
...
I A N b
I A O s
I A SIL e
I B A i
...
```

- Añade la lista de los fonemas CI base al inicio de la lista resultante.

```
A -  -  -
A_1 -  -  -
B -  -  -
...
A CH E e
A D D i
A I L i
A I S i
A L I e
A L S e
A L U b
...
I A N b
I A O s
I A SIL e
I B A i
...
```

  La lista consta de 4 columnas. La primera corresponde a un fonema base, la segunda columna corresponde al fonema que se encuentra en el contexto

izquierdo del fonemas base, la tercera columna corresponde al fonema que se encuentra en el contexto derecho del fonema base y la última indica el tipo de trifonema de que se trata: `b` indica que el trifonema está al inicio de la palabra, `e` indica que el trifonema está al final de la palabra, `i` indica que el trifonema está en medio de la palabra (word internal triphone) y `s` indica que se trata un único trifonema. El atributo `-` indica que se trata de fonemas base por lo que no se encuentran en ningún contexto mencionado.

- La lista de fonemas y trifonemas anterior se convierte al archivo de definición del modelo que lista a *todos* los posibles trifonemas a partir del diccionario.

Finalmente, se construye un *archivo de definición del modelo que incluye solamente una lista corta de trifonemas y fonemas CI* que hayan sido observados. Esta lista define al *archivo de definición del modelo no ligado CD* final, que será utilizado para el entrenamiento de los modelos no ligados CD. Al igual que el archivo de definición del modelo CI, el archivo de definición del modelo no ligado CD asigna un único identificador a cada estado HMM y servirá como un archivo de referencia para manejar e identificar a los parámetros del modelo no ligado CD. De nueva cuenta se hace uso de la aplicación `mk_mdef_gen`, que esta vez realizará los siguientes pasos:

- Encontrar el número de veces que cada uno de los trifonemas, listados en el procedimiento anterior, ocurre en el corpus de entrenamiento. Por ejemplo, el siguiente listado muestra un segmento de la ocurrencia de varios fonemas y trifonemas que se dan en este trabajo.

```
A - - - 85
A_1 - - - 3
B - - - 15
...
A CH E e 4
A D D i 4
A I L i 3
A I S i 9
A L I e 3
A L S e 4
A L U b 4
...
I A N b 6
I A O s 9
I A SIL e 4
I B A i 3
...
```

El número final en cada renglón muestra el número de veces que cada trifonema o fonema de relleno particular que ha ocurrido en el corpus de entrenamiento. Sin embargo, existe la posibilidad de que si todos los posibles trifonemas de un fonema CI son listados en el archivo de todos los trifonemas, el fonema CI en sí mismo tenga una cuenta de cero, puesto que todas sus instancias podrían haber sido relacionadas con un trifonema.

- La lista de los trifonemas contados se usa para componer una lista más corta de trifonemas que hayan ocurrido un mínimo número de veces, es decir, que se encuentren dentro de un umbral. Esta lista corta aparece en el mismo formato que el formato del archivo a partir del cuál hayan sido seleccionados. La lista corta de trifonemas que se genere tendrá el mismo formato que el de la lista de los trifonemas usados en el archivo de todos los trifonemas. Como antes también se incluye la lista de fonemas CI. Así, por ejemplo, si se tuviera un umbral de 4, la lista anterior quedaría como:

```
A - - - 85
B - - - 15
...
A CH E e 4
A D D i 4
A I S i 9
A L S e 4
A L U b 4
...
I A N b 6
I A O s 9
I A SIL e 4
...
```

  El umbral mencionado se debe ajustar de tal manera que el número total de trifonemas que se encuentran por arriba del umbral sea menor que el número máximo de trifonemas que el sistema pueda entrenar o que se desee entrenar. Es recomendable entrenar tantos trifonemas como sea posible. Sin embargo, el número de trifonemas puede ser dependiente de la memoria disponible de la máquina. Se busca, entonces, que el umbral sea establecido de tal manera que los modelos resultantes sean lo suficientemente pequeños para ocupar poca memoria. En caso de que la memoria no sea problema entonces se ajusta el umbral para que sea un número muy pequeño.
  Si el umbral es demasiado pequeño, puede ocasionar que el trifonema ocurra pocas veces, lo que puede generar insuficiencia de datos a la hora de querer entrenar las distribuciones de probabilidad de los estados HMM propiamente. Lo que conduce a que los modelos no ligados dependientes del contexto sean pobremente estimados, que, a su vez, afectarían los árboles de decisión que usan estos modelos para ser construidos en la siguiente etapa decisiva del entrenamiento.

- Se crea un archivo de definición del modelo que incluya solamente la lista corta de trifonemas. Este es el archivo final que será utilizada para el entrenamiento no ligado CD.

5.1.6. Inicialización plana de los parámetros no ligados dependientes del contexto

En cuanto se haya creado el archivo de definición del modelo no ligado CD, el siguiente paso en el entrenamiento no ligado CD es la etapa de *inicialización de los parámetros no ligados dependientes del contexto*. Durante esta etapa se

generan los archivos de los parámetros del modelo que le corresponden al archivo de definición del modelo no ligado CD. Tales archivos son los archivos de las medias, de las variancias, de las matrices de transición y de los pesos mixtos de los modelos no ligados CD. Para asignarles valores a cada uno de estos archivos, primero se toman sus respectivos archivos de parámetros de los modelos CI y se copian los valores de cada uno de ellos a sus correspondientes archivos de modelos no ligados CD. Cada estado de un fonema CI en particular, contribuye al estado que tenga el mismo fonema CI en el archivo de parámetros no ligados CD y también contribuye al mismo estado de todos los trifonemas que contengan a ese fonema CI y que se encuentren en el archivo de parámetros no ligados CD. El archivo de definición del modelo no ligado CD es, por su puesto, utilizado como referencia para establecer esta relación.

La inicialización para el entrenamiento no ligado CD se lleva a cabo mediante la aplicación `init_mixw`, cuyos argumentos se describen en la tabla E.13 del apéndice E. Este software realiza los siguientes procedimientos:

- Realiza la lectura de los archivos de parámetros del entrenamiento CI: el archivo de definición del modelo CI y los archivos de los pesos mixtos, las matrices de transición, las media y las variancias del modelo CI.
- Realiza la lectura del archivo de definición del modelo no ligado CD (CD untied model file).
- Cuando se trata de los fonemas base, copia todos los parámetros que corresponden a cada estado de cada fonema CI a los respectivos estados del modelo no ligado CD. Por ejemplo, la matriz de transición, las medias, la variancias y los pesos mixtos del fonema `A` del modelo ligado CI, son copiados a los mismos parámetros del mismo fonema `A` en el modelo no ligado CD.
- Cuando se trata de los trifonemas, la inicialización de los estados de cada trifonema se hace con respecto a sus fonemas base que son iguales a los fonemas del modelo CI. Por ejemplo, el trifonema `A CH E e`, tiene a `A` como fonema base; por lo tanto este trifonema se inicializa con los parámetros del mismo fonema que se toma del modelo CI. Si por alguna razón el fonema base no existiera en los fonemas del modelo CI, el trifonema sería inicializado uniformemente.

5.1.7. Entrenamiento de los modelos no ligados dependientes del contexto

Lo que prosigue a la inicialización de los parámetros del entrenamiento no ligado CD es, de hecho, entrenar los modelos no ligados CD. Para llevar a cabo ésto se utiliza de nueva cuenta el algoritmo Baum-Welch en forma similar a como se hizo en el entrenamiento CI[1]. Así, de nuevo, se iteran las aplicaciones `bw` y `norm` hasta alcanzar un criterio de convergencia. Cada iteración genera los

[1] Véase la sección 5.1.4

parámetros estimados, es decir, las medias, las variancias, los pesos mixtos y las matrices de transición, nada más que, ahora para los modelos no ligados dependientes del contexto. En modelos no ligados dependientes del contexto el criterio de convergencia típicamente sucede en el rango de 6 a 10 iteraciones.

5.1.8. Construcción de los árboles de decisión para compartir parámetros

El siguiente gran paso en el entrenamiento de modelos continuos es la *construcción de los árboles de decisión*. Los árboles de decisión son utilizados para elegir a aquellos estados HMM de todos los trifonemas (vistos y no vistos) que sean similares a algún otro estado, de manera tal que todos los datos de estos estados son reunidos y utilizados para entrenar a un estado global o senón. En este procedimiento se forman muchos grupos de estados similares y el número de senones que se entrenan finalmente puede ser definido por el usuario.

Generación de las preguntas lingüísticas

Para poder construir los árboles de decisión es necesario contar con los modelos no ligados CD y con un conjunto de clases fonéticas predefinidas, que corresponden a las clases de las unidades acústicas que hayan sido modeladas y que comparten alguna propiedad en común. Dichas clases se denominan también *preguntas* y se utilizan para realizar particiones de los datos en cualquier nodo dado de un árbol. Cada pregunta genera una partición. Para particionar los datos en un nodo determinado se elige a aquella pregunta en la que se obtenga el máximo aumento en la probabilidad debido a la partición, es decir, aquella que resulte con la "mejor" partición.

Todas las preguntas lingüísticas se guardan en un solo archivo llamado el *archivo de preguntas lingüísticas* (*linguistic questions file*). Se debe construir un árbol de decisión para cada estado de cada fono.
Un ejemplo de un archivo de preguntas lingüísticas es el siguiente:

```
SIL  SIL
VOCALES A E I O U
ALVEOL  N L R RR S
DENTAL  D T N L
LABDEN  F M
LIQUID  L R
```

Obsérvese que consta de dos columnas. La columna de la izquierda especifica el nombre dado a cada clase. La columna de la derecha corresponde al fonema o al grupo de fonemas que compartan alguna propiedad acústica en común. El nombre dado a una clase puede ser definido por el usuario o ser asignado en forma automática. En el caso de que las unidades acústicas no se basen completamente en el aspecto fonético ó si se están entrenando modelos de un lenguaje para el cuál no se tiene certeza acerca de sus estructuras fonéticas, entonces es posible hacer uso de la aplicación de clasificación, denominada `make_quests`, que se proporciona en *Sphinx 3*. Dicha aplicación se utiliza para

generar las preguntas lingüísticas a partir de los parámetros de los modelos CI y sus argumentos se detallarán en la tabla E.7 del apéndice E. A grandes rasgos `make_quests` genera las preguntas lingüísticas de la siguiente manera: Para cada árbol que corresponde a la posición, de un estado dentro del modelo acústico, se barre el árbol de arriba hacia abajo, y de izquierda a derecha. Para cada nodo, se genera una lista ordenada con los fonemas existentes en el nodo y estos son agregados al archivo de preguntas lingüísticas [29].

Construcción de los árboles de decisión

Después de que las preguntas lingüísticas hayan sido generadas, se deben construir los árboles de decisión; uno para cada estado de cada fonema CI que se encuentre presente en la lista de los fonemas de entrenamiento. Por ejemplo, si se tuvieran 3 fonemas de entrenamiento y cada fonema estuviera representado con HMMs de tres estados, se tendrían 9 árboles de decisión. Sin embargo, los árboles de decisión no se construyen para los fonemas de relleno ni tampoco para el fonema *SIL*. Para la construcción de los árboles, se usa el ejecutable `bldtreee`. En la tabla E.8 del apéndice E se describen sus argumentos.

Podado de los árboles de decisión

Después de que los árboles de decisión hayan sido creados se procede a podarlos para que tengan tantas hojas como el número de estados ligados que se desee entrenar. Cabe aclarar que el número de estados ligados no incluyen a los estados CI y que estos últimos nunca son ligados. Durante el proceso de podado se remueven progresivamente aquellas bifurcaciones de los árboles de decisión que hayan obtenido un incremento mínimo en su probabilidad reemplazándolas por su nodo padre. La selección de las ramas que serán recortadas se hace yendo en forma global a través de la colección entera de árboles de decisión. El ejecutable que se usa para recortar los árboles de decisión se denomina `prunetree`, cuyos argumentos se encuentran en la tabla E.10 del apéndice E.

5.1.9. Creación del archivo de definición del modelo ligado CD

Después de realizar el podado de los árboles, es necesario crear un nuevo archivo de definición del modelo CD que contemple lo siguiente:

- Contenga a todos los trifonemas vistos en el entrenamiento.
- Tenga a los estados correspondientes a esos trifonemas identificados con los senones de los árboles de decisión.

Para ello, se utiliza el archivo de definición del modelo que contiene a todos los posibles trifonemas del diccionario de entrenamiento actual (alltriphone

model definition file), el cuál fue creado durante el proceso de construcción del archivo de definición del modelo no ligado dependiente del contexto, como se vió en la sección 5.1.5. Recuerde que el archivo de definición del modelo no ligado dependiente del contexto (CD) contiene solamente un número seleccionado de trifonemas con varios umbrales que fueron usados para la selección. Razón por la que este último no puede ser usado para construir el archivo de definición del modelo ligado CD.

La aplicación que se utiliza para ligar los estados se denomina `tiestate` y opera con los argumentos que se detallan en la tabla E.9 del apéndice E. A grandes rasgos, el software `tiestate` realiza los siguientes pasos:

- Realiza la lectura del archivo de definición del modelo con todos los posibles trifonemas en el entrenamiento.
- Para cada estado de los fonemas base se realiza la lectura de su árbol de decisión correspondiente y se etiquetan sus hojas con un identificador único o ID. El valor de dicho ID consiste de un número entero que es asignado a partir del último identificador de los estados de los fonemas base mas uno. Por ejemplo, si se tienen 22 fonemas base y se modela cada uno con 3 estados HMM, obtenemos 66 estados para estos fonemas base; cada uno con su respectivo ID, iniciando en 0 y terminando en 21. Los ID's que le corresponden a los senones (hojas de los árboles) comenzarán a partir del número 22 y terminanarán hasta que se haya alcanzado el total de hojas de todos los árboles.
- Se crea el encabezado del archivo de definición del modelo ligado CD de salida, dado por el parámetro `omoddeffn` , que se obtiene a partir del archivo de definición del modelo no ligado CD de entrada, dado por el parámetro `imoddeffn` de la siguiente forma:

  `omoddeffn->n_base = imoddeffn->n_base`
  `omoddeffn->n_tri = imoddeffn->n_tri`
  `omoddeffn->n_state_map = imoddeffn->n_state_map`
  `omoddeffn->n_tied_state = imoddeffn->n_tied_ci_state + n_senone`
  `omoddeffn->n_tied_ci_state = imoddeffn->n_tied_ci_state`
  `omoddeffn->n_tied_tmat = imoddeffn->n_tied_tmat`

  Observe que se realiza una copia del encabezado del archivo de definición del modelo no ligado CD de entrada al archivo de definición del modelo ligado CD de salida. La diferencia entre encabezados está dada por el número de estados ligados (`n_tied_state`), que se calcula por la suma de los estados en los fonemas base (`n_tied_ci_state`) más el número de senones que haya (`n_senone`).
- Se definen los ID's con respecto a cada estado en los modelos de la siguiente forma:
  En el caso de los fonemas base los ID's de los estados se copian desde el archivo de definición del modelo de entrada al al archivo de definición del modelo de salida.

Para la obtención de los ID's de los trifonemas en el archivo de definición de salida, se realiza para cada estado de los trifonemas del archivo de definición del modelo de entrada lo siguiente: se lee el árbol de decisión correspondiente a su fonema base y a la posición del estado en el modelo, por ejemplo, para cuando se modelan trifonemas con tres estados, la posición del estado en el modelo puede ser 0, 1 ó 2. A partir del nodo raíz, en el árbol de decisión, se evalúan las preguntas compuestas usando el trifonema dado, se recorre el árbol hasta llegar a un nodo hoja. El identificador único (ID) de esta hoja se usa para establecer el ID del estado correspondiente al trifonema en el archivo de salida.

5.1.10. Inicialización y entrenamiento de los modelos ligados CD con Gaussianas mixtas

El paso siguiente es el *entrenamiento de los modelos ligados CD* (*CD-tied models*). Cuando se trata de modelos continuos, los estados HMM pueden ser modelados por una sola distribución Gaussiana o por varias distribuciones Gaussianas mixtas. Con el fin de tener pocos problemas de convergencia, se recomienda que el número de Gaussianas en una distribución mixta sea par y potencia de dos (2, 4, 8, 16, 32, etc.), aunque ésto no es necesario, como se observa en la figura 5.8 para 15 Gaussianas. El proceso de entrenamiento de los modelos ligados CD con N Gaussianas mixtas se puede observar en la última parte de la figura 5.2.
Por ejemplo, si se quisiera modelar los estados HMM con una mezcla de 8 Gaussianas, primero es necesario inicializar y entrenar con 1 sola Gaussiana por estado de los modelos HMM. A partir de ella, cada distribución Gaussiana es dividida en dos, haciendo una perturbación ligera en su media, como se observa en la figura 5.9a. Las dos distribuciones Gaussianas resultantes se utilizan para inicializar y realizar el entrenamiento de modelos ligados CD con 2 Gaussianas por estado en los modelos HMM. Nuevamente, se toman esas dos densidades Gaussianas y se llevan a cabo sendas perturbaciones ligeramente en sus respectivas medias, como se observa en la figura 5.9b. De nueva cuenta las 4 Gaussianas resultantes se utilizan para inicializar y realizar el entrenamiento de 4 Gaussianas por estado en los modelos HMM. El mismo procedimiento se lleva a cabo para 8 Gaussianas por estado en los modelos HMM. Así el entrenamiento ligado CD para modelos con 2^N Gaussianas por estado se hace en $N + 1$ pasos. Cada uno de esos pasos consiste en:

1. Inicialización.
2. Iteración con el algoritmo Baum-Welch seguido de su respectiva normalización.
3. Partición de Gaussianas.

Obsérvese que el procedimiento terminará en $N + 1$ pasos, es decir, cuando se hayan alcanzado el número de Gaussianas desesado. Sin embargo, éstas últimas

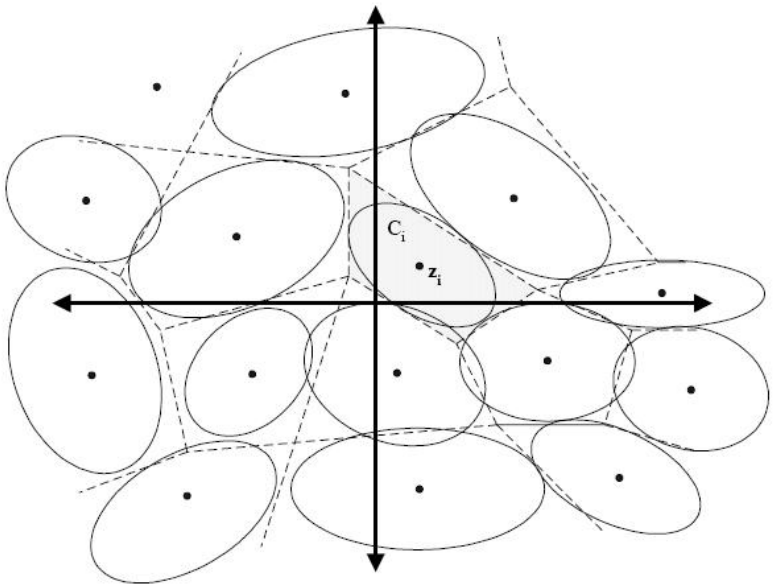
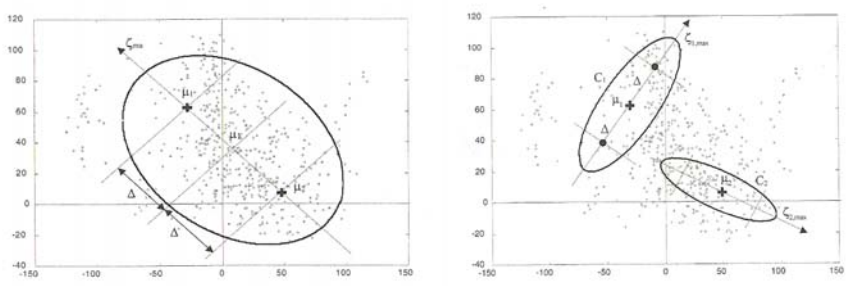


Figura 5.8: Ejemplo de obtención de 15 densidades Gaussianas [15].



(a) (b)

Figura 5.9: (a) División de 1 densidad Gaussiana y (b) División de densidades Gaussianas en la siguiente iteración [3].

también se necesitan inicializar y entrenar (Pasos 1 y 2) pero no dividir, por lo que el paso 3 no se aplicará en el paso $N + 1$.

El entrenamiento comienza con la inicialización de 1 Gaussiana por estado en los modelos HMM. Durante la etapa de inicialización, los parámetros del modelo en los archivos de parámetros del modelo CI, se copian en localidades apropiadas de los archivos de parámetros del modelo ligado CD. Como en cada entrenamiento, se crean cuatro archivos de parámetros, uno para la media, otro para la variancia, el tercero para las matrices de transición y el último para los pesos mixtos. Durante la inicialización, cada estado de un fonema CI, en particular, contribuye al mismo estado del fonema CI en el archivo de definición del modelo ligado CD, también para el mismo estado de todos los trifonemas del mismo fonema CI en el archivo de parámetros del modelo ligado CD. Se utiliza el archivo de definición del modelo ligado CD como referencia en estas relaciones.

Inicialización plana de los modelos ligados CD para N Gaussianas

Aquí, la inicialización se realizará ya sea para 1 Gaussiana por estado o para N Gaussianas por estado en los modelos ligados CD, dependiendo si se trata de la primera iteración o si se trata de las siguientes N iteraciones. La inicialización se lleva a cabo mediante el ejecutable llamado `init_mixw`, cuyos argumentos se muestran en la tabla E.13 del apéndice E.
Durante la primera iteración, la aplicación `init_mixw` realiza los siguientes procedimientos:

- Realiza la lectura de los archivos correspondientes al entrenamiento CI: se leen el archivo de definición del modelo CI, el archivo de pesos mixtos del modelo CI, el archivo de las medias correspondientes al modelo CI, el archivo de las variancias del modelo CI y el archivo de las matrices de transición correspondientes al modelo CI.

- Realiza la lectura del archivo de definición del modelo ligado CD. Son necesarios dos archivos de definición del modelo, uno sin trifonemas para la definición del modelo CI y otro que contenga a los trifonemas para la definición de trifonemas del modelo CD, ya que la meta es entrenar por etapas, primero los modelos CI y luego los modelos CD. Todo esto para minimizar los problemas de inicialización.

- Se copian los parámetros correspondientes a los fonemas base de los modelos CI, es decir, matriz de transición, medias, variancias y pesos mixtos, a los correspondientes fonemas base de los modelos ligados CD.

- Para los trifonemas, se obtienen los parámetros con respecto a los de sus fonemas base, es decir, para cada trifonema en el modelo ligado CD, se inicializan sus parámetros utilizando los respectivos valores de su fonema base en el modelo CI. Por ejemplo, el trifonema `A CH E e`, tiene como fonema base `A,`por tanto este trifonema se inicializa con los parámetros del

mismo fonema `A` en el modelo CI. Si por alguna razón el fonema base no existiera en los fonemas del modelo CI, éste será inicializado uniformemente.

En las siguientes N iteraciones, `init_mixw` realiza lo siguiente:

- Se realiza la lectura del archivo de definición del modelo ligado CD, del archivo de pesos mixtos del modelo ligado CD, del archivo de las medias correspondientes al modelo ligado CD, del archivo de las variancias del modelo ligado CD y del archivo de las matrices de transición correspondientes al modelo ligado CD.
- Realiza copias de los archivos anteriores para preservarlas.
- Realiza la lectura del archivo de definición del modelo ligado CD.
- Se copian los pesos mixtos, las medias, las variancias y las matrices de transición desde un modelo CD a otro modelo CD que tiene el número de densidades Gaussianas incrementado. Las matrices de transición no cambian por lo que son copiadas íntegramente. Se copian los parámetros correspondientes a los fonemas base de los modelos CD, a los correspondientes fonemas base de los modelos ligados CD que tienen el número de densidades Gaussianas incrementado.
- Para el caso de los trifonemas, se obtienen los parámetros con respecto al de sus fonemas base, es decir, para cada trifonema en el modelo ligado CD con el número de Gaussianas incrementado, se inicializa sus parámetros usando los valores correspondientes a su fonema base en el modelo CD. Si por cualquier situación el fonema base no existe en el los fonemas del modelo CD, este será inicializado uniformemente.

Entrenamiento de los modelos ligados CD con Gaussianas mixtas

Después de haber inicializado los parámetros de los modelos ligados CD con N Gaussianas, el siguiente paso consiste en entrenar estos modelos. Para llevar a cabo ésto es necesario iterar el algoritmo Baum-Welch, realizando también su respectiva normalización. Para calcular los parámetros finales de cada iteración es necesario que se ejecuten las aplicaciones `bw` y `norm` en el cuerpo de entrenamiento. Los argumentos necesarios para la aplicación `bw` se dan en las tablas E.15 y E.16 del apéndice E y los argumentos para la aplicación `norm` se dan en la tabla E.14 también del apéndice E. De nueva cuenta, las iteraciones terminan hasta que la probabilidad converja, lo cuál ocurre, generalmente, entre 6 y 10 iteraciones.

Debido a que las ejecuciones de `bw` y `norm` son muy similares a los realizados en el caso de CI, no se describen en detalle.

Incremento del número de gaussianas en los modelos ligados CD

Una vez que ha convergido el entrenamiento de los modelo ligados CD para N Gaussianas, el paso que sigue es la *división de las Gaussianas*, es decir la creación de dos nuevas Gaussianas por cada Gaussiana que exista actualmente. Para realizar la partición de las densidades, se usa el ejecutable `inc_comp` con los argumentos listados en la tabla E.4 del apéndice E.

La aplicación `inc_comp` realiza los siguientes procedimientos:

- De los archivos correspondientes al entrenamiento ligado CD, solamente realiza la lectura del archivo de pesos mixtos de los modelos CD, del archivo de las medias correspondientes a los modelos CD y del archivo de las variancias de los modelos CD.
- Para todas las mezclas:
  - Encuentra la densidad que tenga el componente más largo que no haya sido dividido, es decir, aquella que sea más probable o que tenga el mayor número de ocurrencias.
  - Divide en dos componentes hasta llegar a $n\_densidades\_actuales + ninc$ densidades de los elementos de los vectores de características. Cada nuevo componente tiene:

    ```
    mixw_a = mixw_b = 1/2 mixw
    mean_a = mean + 0.2 std
    mean_b = mean - 0.2 std
    var_a = var_b = var
    ```

    donde `mixw_a` y `mixw_b` son los nuevos pesos mixtos y `mix_w` es el peso mixto actual; `mean_a` y `mean_b`son las nuevas medias y `mean`es la media actual; `var_a`, `var_b`son las nuevas variancias y `var`es la variancia actual.
- Copia los nuevos pesos mixtos, las nuevas medias y las nuevas variancias a los respectivos archivos de salida del modelo CD.

5.2. Visión general del decodificador de *Sphinx*

El decodificador $s3.8$ de *Sphinx 3* es la implementación más reciente para reconocimiento de voz a texto. Sin embargo, en este trabajo se utilizó el decodificador $s3.6$ por dos razones: la primera porque en el tiempo en que comencé a utilizarlo era la versión más reciente en ese entonces, y la segunda, porque dicho decodificador tuvo un incrementó en diez veces más que la versión anterior ($s3.4$) en tareas de grandes vocabularios [38]. El decodificador incluye el siguiente software [6]:

- `sphinx3_decode`: El decodificador de *Sphinx 3* $s3.2/s3.3/s3.X$ para procesar archivos de tipo cepstral. También es llamado `s3 fast`, es un

decodificador en modo intérprete de comandos (batch) que utiliza vocabulario complejo (tree lexicon), con optimización tanto en modelos de mezclas Gaussianas como en validación transversal de palabras cruzadas de trifonemas. Su precisión es menor que `sphinx3_decode_anytopo` pero su velocidad es 5 veces mayor a éste. A partir de la versión *s*3.6 también permite decodificación con gramáticas de estados finitos y también permite cambiar el tipo de búsqueda. Por tanto, es un reconocedor bien balanceado y podría ser la primera elección de reconocimiento en modo intérprete de comandos.

- `sphinx3_livedecode:` Es una demostración para pruebas en vivo. que utiliza el motor de `sphinx3_decode`. `sphinx3_livedecode` permite al usuario tratar de reconocer con sólo oprimir un botón. No es una máquina de dictado. Su función es la de mostrar como la API de reconocimiento en tiempo real puede ser usada para construir un sistema.

- `sphinx3_livepretend`: Es un simulador de pruebas en vivo que utiliza el motor de `sphinx3_decode`. `sphinx3_livepretend` segmenta una señal de voz y "alimenta" los segmentos de la señal de voz al motor de búsqueda de `sphinx3_decode`. La segmentación se realiza en forma ad-hoc al seleccionar 25 tramas por segmento. Lo que provoca que su velocidad sea ligeramente menor a `sphinx3_decode`. Los parámetros de esta aplicación se detallan en las tablas F.2 y F.3 del apéndice F.

- `sphinx3_decode_anytopo:` El decodificador de *Sphinx 3* que procesa archivos cepstral (para compatibilidad con versiones anteriores). También llamado `s3 slow` es un decodificador en modo intérprete de comandos que utiliza vocabulario simple o plano (flat) con optimización cruzada de trifonemas y validación de palabras cruzadas. Su desempeño es muy cercano a un reconocedor de trifonemas y de validación de palabras cruzadas completos. Fue utilizado en CMU de 1994 a 2000 y tiene el mejor desempeño de todos los reconocedores de *Sphinx*.

- `sphinx3_align:` Es un ejecutable para realizar alineamiento. Dada la transcripción de una locución, `sphinx3_align` puede proporcionar al usuario el nivel de alineación por estado, fono o palabra.

- `sphinx3_allphone:`Lleva a cabo reconocimiento de fonemas para una locución realizando expansión total de trifonemas.

- `sphinx3_dag:` Tanto `sphinx3_decode` como `sphinx3_decode_anytopo` pueden generar celosías para una segunda etapa de resultados. En forma similar a una máquina `sphinx3_dag` puede obtener estos resultados realizando una búsqueda para la mejor ruta dentro de la celosía dado un modelo del lenguaje basado en trigramas.

- `sphinx3_astar:` Dada una celosía `sphinx3_astar` puede generar una lista del N-mejor en *Sphinx 3*. Es bastante útil para generar los resultados de la búsqueda del N-mejor.

- `sphinx3_continuous:` Ejecutable para una demostración en vivo y para aplicaciones sencillas de habla y reconocimiento.
- `sphinx3_gausubvq:` Para construir subvectores de los clusters de los modelos acústicos.

Para llevar a cabo el reconocimiento en este trabajo se ejecutó la aplicación `sphinx3_livepretend`, cuyos argumentos principales se describen en la tabla F.1 del apéndice F y los parámetros del archivo de configuración se detallan en las tablas F.2 y F.3 del mismo apéndice.

El proceso de evaluación se realizó con la herramienta `sphinx3_align`, cuyos argumentos se listan en las tablas F.4 y F.5 del apéndice F.

El decodificador `s3.6` se basa en el algoritmo de búsqueda Viterbi y como función heurística emplea la búsqueda beam. Además utiliza una estructura de árbol de búsqueda de léxico. Toma como entrada grabaciones de voz en formato raw PCM y escribe los resultados del reconocimiento en archivos de salida.
Brevemente se hará una descripción de los parámetros de entrada del decodificador. Además de los *diccionarios de pronunciación y de relleno*, el decodificador utiliza otros parámetros de entrada para realizar su trabajo. El decodificador requiere de:

- *modelo acústico. Sphinx 3* está basado en modelos acústicos por subfonemas. El modelo acústico se obtuvo como resultado del entrenamiento de los modelos ligados CD con N Gaussianas y se representa por la siguiente colección de archivos: Un *archivo de definición del modelo*, los *archivos de medias y de variancias Gaussianas*, un *archivo de pesos mixtos* y un *archivo de matrices de transición de estados.*
- *modelo del lenguaje.* El modelo de lenguaje (LM) principal que utiliza el decodificador de Sphinx es un modelo de lenguaje convencional basado en bigramas o trigramas. Se puede observar el modelo del lenguaje usado en esta tesis en la sección 5.3.
- *archivo de control.* Este archivo le indica al decodificador los parámetros de comportamiento. El decodificador de *Sphinx 3* procesa las entradas listadas en dicho archivo. Cada línea en el archivo de control identifica una pronunciación separada. Por ejemplo, el archivo de control para el decodificador presenta las siguientes entradas:

```
BeatrizC/DECODE/PARA/BC01
BeatrizC/DECODE/PARA/BC02
BeatrizC/DECODE/PARA/BC03
```

El funcionamiento del reconocedor de *Sphinx* se describe a continuación:

1. **Inicialización**. En esta etapa se configura al decodificador; dicha configuración se mantendrá durante toda la ejecución del mismo. Esta etapa a su vez comprende los siguientes pasos:

   - *Inicialización de la base logarítmica. Sphinx* lleva a cabo todos los cálculos de probabilidades en el dominio logarítmico. Con el fin de mantener la eficiencia computacional, la base logarítmica se elige de tal manera que las probabilidades logarítmicas se mantengan como valores enteros de 32 bits. Así, todos los puntajes reportados por el decodificador son probabilidades logarítmicas en esta base peculiar (1.0003). La base logarítmica puede cambiarse por valores dentro de un rango muy amplio sin afectar el reconocimiento.
   - *Inicialización de los modelos.* Durante esta etapa se cargan los modelos léxicos (diccionarios de pronunciación y de relleno), los modelos acústicos y el modelo del lenguaje que sean especificados en el archivo de configuración. De hecho, el modelo del lenguaje utiliza una estrategia basada en disco[2], así que sólo se carga parcialmente. El conjunto de modelos se utiliza para decodificar todas las oraciones pronunciadas en la entrada.
   - *Efectividad del vocabulario.* Después de que todos los modelos han sido cargados se determina el conjunto de palabras que el decodificador es capaz de reconocer. Primero se encuentra la intersección de las palabras en el modelo del lenguaje y en el diccionario de pronunciación. Enseguida, basados en el diccionario de pronunciación, se incluyen todas las pronunciaciones alternativas del conjunto de intersecciones obtenidas. Finalmente, se incluyen todas las palabras de relleno a partir del diccionario de relleno, pero excluyendo los identificadores de inicio y fin de oración, `<s>` y `</s>`, respectivamente.
   - *Construcción del árbol de léxico.* Al determinar la efectividad del vocabulario, el decodificador construye *árboles de léxico*. Se construyen árboles separados tanto para las palabras del diccionario de pronunciación como del diccionario de relleno, e incluso se pueden hacer varias copias de ambos árboles.

2. **Procesamiento del archivo de control**. Después de la inicialización, el decodificador procesa las entradas en el archivo de control en forma secuencial, una por una. El decodificador carece de capacidad de aprendizaje o de adaptación. `sphinx3_livepretend` se comporta como si las pronunciaiones de los archivos fueran hechas en tiempo de procesamiento, por lo que cambiar el orden de las entradas en el archivo de control puede afectar los resultados individuales, pero puede ser imperceptible si el ambiente en el que se grabaron las locuciones permanece constante.

3. **Podado**. Cada pronunciación o entrada en el archivo de control se procesa usando los modelos dados y usando el algoritmo de búsqueda Viterbi.

[2]Véase el parámetro `lminmemory` de la tabla F.2 del apéndice F

Con el fin de restringir el espacio de búsqueda activo a límites computacionalmente manejables, se realiza el proceso de *podado*, que consiste en descartar a las hipótesis menos prometedoras que se dan durante el proceso de reconocimiento. Existen dos tipos de podado:

- *podado por haz* (*beam prunning*). Cada locución es procesada en forma síncrona en el tiempo, una trama a la vez. En cada trama el decodificador tiene un número de HMMs que se encuentran activos actualmente para que sean relacionados con la siguiente trama de la señal de entrada. Sin embargo, primero se descartan aquellas probabilidades que se encuentren debajo de un *umbral*, el cuál se establece con respecto a la mejor probabilidad de estados HMM en esa trama y se obtiene al multiplicar la mejor probabilidad de estados por un *ancho del haz* (*beamwidth*) fijo.
  Un podado por haz similar se lleva a cabo en varios procesos del decodificador. Por ejemplo, para determinar las palabras candidatas que son reconocidas en cualquier instante o para determinar los componentes de las densidades en una mezcla Gaussiana que están muy cercanas a un vector de características dado.
- el *podado absoluto* (*absolute prunning*. Aún con el podado por haz, el número de entidades activas a veces puede llegar a ser computacionalmente irrefrenable. Si existe un gran número de HMMs que caiga dentro del umbral de podado, el decodificador los mantendrá a todos activos. Sin embargo, si el número de HMMs activos crece más allá de ciertos límites, las oportunidades de detectar la palabra entre todas las candidatas son considerablemente reducidas. Tal situación puede ocurrir, por ejemplo, si la señal de entrada contiene demasiado ruido o no se relaciona en ningún aspecto con los modelos acústicos. En tales casos, no existe razón para permitir que el espacio de búsqueda activo crezca hasta alcanzar extensiones arbitrarias. Puede ser contenido al usar parámetros de podado que limiten el número absoluto de entidades activas en cualquier instante.

4. **Cálculo de GMM en forma rápida** El cálculo de la probabilidad de distribución Gaussiana puede ser uno de los factores dominantes en el cómputo de los GMM (Gaussian Mixture Models) o senones. Los modelos acústicos de densidades continuas son costosos en términos computacionales para tratar con ellos, puesto que contienen cientos de miles de densidades Gaussianas que deben ser evaluadas en cada trama. Para reducir este costo, se puede utilizar un modelo aproximado que identifique eficientemente aquellas densidades candidatas que tengan el mayor puntaje en cada mezcla Gaussiana de cualquier trama. Así, las densidades restantes se ignoran durante esa trama. En *Sphinx 3* tal modelo de aproximación se construye al *cuantizar por sub-vectores* las densidades del modelo acústico.
   Durante el entrenamiento se calcula con anterioridad un conjunto de code-

books cuantizados vectorialmente (VQ codebook) para todas las medias. Se calculan los vecinos para cada senón de las palabras código (codeword) sólo si la media de una Gaussiana está muy cerca de la palabra código. Durante el reconocimiento se trata de hallar la palabra código más cercana para esta característica y se calculan las distribuciones Gaussianas solo cuando se encuentran los vecinos con el criterio anterior.

5. **Generación de salidas** Durante el reconocimiento, el decodificador construye una *tabla interna de apuntadores de retroceso* (*backpointer table* o *BP*) a partir de la cuál se generan las salidas finales. Esta tabla registra todas las palabras candidatas que son reconocidas durante la decodificación junto con sus atributos, tales como el tiempo de segmentación, sus probabilidades acústica y del lenguaje. Asimismo registra las entradas de sus predecesores en la tabla.

   Cuando una locución ha sido completamente procesada se extrae la mejor hipótesis de reconocimiento de la siguiente manera: En esencia, un reconocedor de voz basado en HMM simplemente trata de componer un grafo de estados y busca la mejor ruta dentro del grafo, como se mencionó en el capítulo 4. Por tanto, para el primer nivel se requiere de un grafo de palabras. En este caso un $3-$grama. Pero podría ser una gramática de estados finitos.
   Para el segundo nivel, el reconocedor necesita saber de qué manera se relacionará una palabra con una secuencia de fonemas. Esta es la justificación del uso del diccionario. Como la mayoría de los reconocedores de gran vocabulario *Sphinx* también utiliza conceptos de relleno. La razón principal para ello es que si se especificaran los rellenos en el modelo del lenguaje no resultaría viable y se requiere de tratamiento especial para los elementos de relleno. Esta es la razón por la que se manejan en su propio diccionario.
   Para el nivel final, el reconocedor necesitará saber de qué manera se representa cada fonema como una secuencia de estados. Esta es la razón por la que el conjunto HMM fue utilizado. Este procedimiento se muestra en la figura 5.10.

   También es posible que la tabla sea convertida en una *celosía de palabras* (*Word lattice*) y sea almacenada en un archivo de salida. A continuación se detallarán las salidas del decodicador de *Sphinx 3*.

   a) *Salida de Hipótesis de reconocimiento.* El decodificador de *Sphinx 3* produce una hipótesis de reconocimiento para cada pronunciación que procesa. Las hipótesis para todas las pronunciaciones que se procesan en un sola corrida se escriben en un único archivo de salida, usando una línea para cada pronunciación. El formato de cada línea se presenta como:

$$u\ \mathtt{S}\ s\ \mathtt{T}\ t\ \mathtt{A}\ a\ \mathtt{L}\ l\ sf\ wa\ wl\ wd\ sf\ wa\ sl\ wd\ \ldots\ nf$$

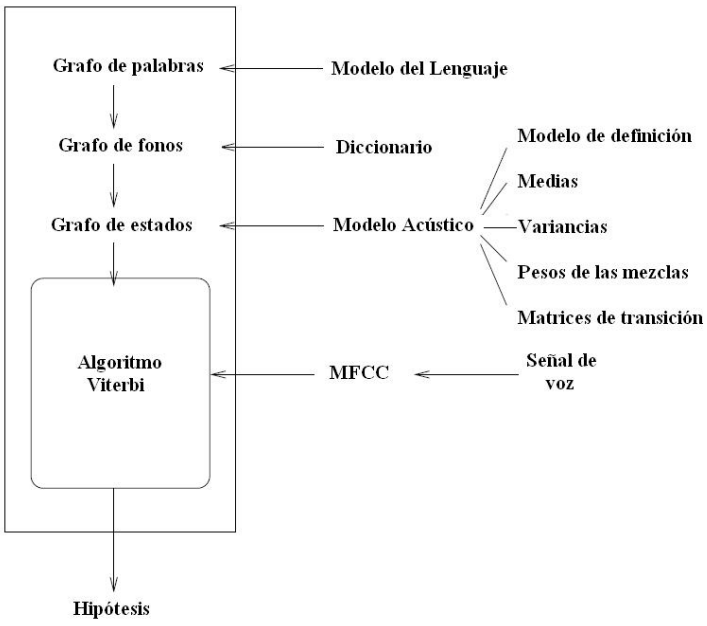


Figura 5.10: Esquema del decodificador de *Sphinx* [6].

Los campos `S`, `T`, `A` y `L` son palabras clave que aparecen en la salida y que se describen a continuación:

u : el identificador de pronunciación

s : un valor acústico escalado que se obtiene durante el cálculo de la probabilidad acústica.

t : el resultado total para esta hipótesis

a : el resultado acústico total para esta hipótesis

l : el resultado del modelo del lenguaje total para esta hipótesis

El campo `l` es seguido por grupos de cuatro campos, un grupo para cada palabra sucesiva en la hipótesis de salida. Los cuatro campos son:

- `sf` : Trama de inicio para la palabra. Su trama final sólo es la trama inicial de la siguiente palabra.
- `wa` : Resultado acústico para la palabra
- `wl` : Resultado del modelo del lenguaje para la palabra.
- `wd` : La palabra en sí.

El campo `nf` es el campo final. En cada línea que corresponde a una hipótesis representa el número total de tramas en la pronunciación. Notese, en primer lugar que todos los resultados son valores logarítmicos de probabilidades en una base en particular que se utiliza en el

decodificador. En segundo lugar, los resultados acústicos son valores escalados; en cada trama, los resultados de los modelos acústicos de todos los senones activos se encuentran escalados de tal manera que el mejor senon tenga una probabilidad logarítmica de 0. Finalmente, los resultados del modelo del lenguaje incluyen los parámetros de peso del lenguaje y de penalización por inserción de palabra.
Como ejemplo se presenta el siguiente fragmento del archivo de las hipótesis de salida para las frases empleadas en este trabajo:

```
WAV/BEATRIZC/DECODE/PARA/BC01F S -3640632 T -7793948 A -7796157 L 2209 0
 -97129 -8101 <SIL> 9 -1158854 -8933 PARA 31 -2254812 -3121 OCHENTA 73 -454950 -
540 Y 76 -683688 -370 OCHO 93 -2225707 -370 PUNTO 130 -2026442 -370 NUEVE 155 -1
973148 -370 NOTICIAS 205 -393738 -8101 <SIL> 226 0 -22938 </S> 226
WAV/BEATRIZC/DECODE/PARA/BC02F S 9101085 T -735219 A -737428 L 2209 0 16
2823 -8101 <SIL> 8 737720 -8933 PARA 28 869223 -3121 OCHENTA 68 239180 -540 Y 72
 533909 -370 OCHO 86 931812 -370 PUNTO 123 1717440 -370 NUEVE 150 2423133 -370 N
OTICIAS 200 250199 -8101 <SIL> 222 0 -22938 </S> 222
WAV/BEATRIZC/DECODE/PARA/BC03F S 5216966 T -662136 A -664345 L 2209 0 14
1042 -8101 <SIL> 8 486526 -8933 PARA 34 421051 -3121 OCHENTA 75 171511 -540 Y 78
 598864 -370 OCHO 97 658732 -370 PUNTO 129 712599 -370 NUEVE 152 969271 -370 NOT
ICIAS 201 8906 -8101 <SIL> 206 0 -22938 </S> 206
```

b) *Salida de celosía de palabras (Word lattice).* Durante el reconocimiento el decodificador mantiene no sólo la mejor hipótesis, sino también un cierto número de alternativas o candidatas, como se vió en la sección 4.5 del capítulo 4. El decodificador opcionalmente puede producir una salida de *celosía de palabras* para cada pronunciación de entrada. Esta salida registra todas las palabras reconocidas por el decodificador que son candidatas en cualquier instante de tiempo así como sus principales atributos tales como la segmentación en el tiempo y los resultados de la probablidad acústica.
El término "lattice" en *Sphinx* no se utiliza en sentido estricto. La celosía de palabras es en realidad un *grafo dirigido acíclico* o *DAG*, donde cada nodo del DAG denota una instancia de una palabra que comienza en una trama en particular dentro de la pronunciación, es decir, que se forma un único par $<$ *palabra*, *tiempo de inicio* $>$. Existe un borde directo entre dos nodos en el DAG si el tiempo de inicio del nodo destino sigue inmediatamente a uno de los tiempo finales del nodo fuente. Esto es, los dos nodos pueden ser adyacentes en el tiempo. Así, el borde termina una posible segmentación para el nodo fuente: comenzando en el tiempo de inicio de la fuente y terminando una trama antes del destino del tiempo de inicio. El borde también contiene una probabilidad acústica para esta segmentación particular del nodo fuente.
Nótese que los tokens de inicio y fin de oración. $< s >$ y $< /s >$, no son decodificados como parte de una pronunciación por el decodificador de Sphinx. Sin embargo, tienen que ser incluidos en el archivo de palabras cruzadas por compatibilidad con versiones anteriores del decodificador. El papel de estos tokens no tiene efecto en el vocabulario y ninguna parte de la señal de entrada es decodificada como alguno de ellos; son solamente indicadores de los límites de las

locuciones y proporcionan el contexto al modelo del lenguaje. Se les asigna en segmentaciones de una trama con resultados de probabilidad logarítmica de 0. Para acomodarlas, las segmentaciones de nodos adyacentes tienen que ser insertados al final en una trama.

5.3. Herramientas para el modelado del lenguaje en *Sphinx*

El *juego de herramientas de Modelado de lenguaje estadístico de Carnegie Mellon* (*The Carnegie Mellon Statistical Language Modeling Toolkit*) o *CMU SLM* es un conjunto de herramientas de software para Linux y está diseñado para facilitar el trabajo del modelado del lenguaje de la comunidad de investigación acerca de Procesamiento de voz.
Algunas de las herramientas se utlizan para procesamiento general de textos como:

- listas de frecuencias de palabras y vocabularios
- conteo de bigramas y trigramas de palabras en general
- conteo de bigramas y trigramas de palabras de vocabularios específicos
- estadísticas relaciondas con bigramas y trigramas
- varios Backoff de modelos de bigramas y trigramas

Los resultados del cálculo del modelo del lenguaje se utilizan para:

- perplejidad
- tasa de palabras fuera de vocabulario
- proporciones de bigramas y trigramas
- anotación de datos de prueba con marcadores de lenguaje

En la figura 5.11 se puede observar la obtención del modelo del lenguaje dado un corpus textual en un archivo sin vocabulario específico. El procedimiento se puede enumerar de la siguiente forma:

1. Calcula el conteo de unigramas con `text2wfreq` . Por ejemplo:

   ```
   bin\text2wfreq <etc/Tesis_trainsin.transcription> etc/Tesis_train.wfreq
   ```

2. Convierte el conteo de unigramas en un vocabulario que consista de n palabras comúnes usando `wfreq2vocab` . Por ejemplo:

   ```
   bin\wfreq2vocab <etc/Tesis_train.wfreq> etc/Tesis_train.vocab
   ```

3. Genera un identificador de trigramas ($id - 3gram$) del texto de entrenamiento, basado en el vocabulario. Utiliza `text2idngram` . Por ejemplo:

```
bin\text2idngram -vocab etc/Tesis_train.vocab <etc/Tesis_trainsin.transcription>
etc/Tesis_train.idngram -write_ascii
```

4. Convierte el identificador de trigramas en un modelo del lenguaje en formato binario o ASCII mediante `idngram2lm` . Por ejemplo:

```
bin\idngram2lm -idngram etc/Tesis_train.idngram -vocab etc/Tesis_train.vocab
-arpa etc/Tesis_train.lm.arpa -context etc/Tesiscontext.css -vocab_type 2 -ascii_input
```

5. Calcula la perplejidad del lenguaje con respecto a algún texto de prueba mediante `evallm` . Por ejemplo: Teclear

```
bin\evallm -arpa etc/Tesis_train.lm.arpa
```

Después teclear

```
perplexity -text etc/Tesis_test.transcription
```

para obtener la perplejidad del lenguaje.

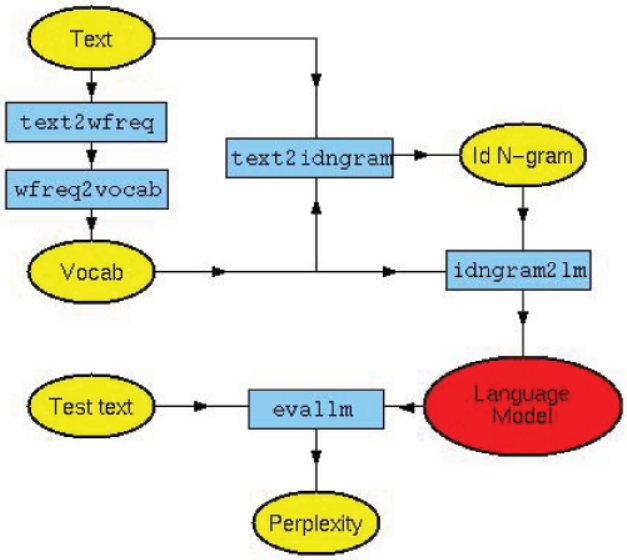


Figura 5.11: Proceso de obtención del modelo del lenguaje con CMU SLM [8].

La gramática generada en el modelo del lenguaje se puede obtener ya sea en formato binario o en formato *ARPA*. Ambos son el mismo modelo del lenguaje nada más que el formato binario es que es usado por `sphix3_decode` y el formato *ARPA* es un formato que se encuentra en código ASCII, para ser analizado por una persona. A continuación se muestra un segmento del archivo del modelo del lenguaje, en formato *ARPA*, utilizado en el presente trabajo para el reconocimiento de frases:

```
\data\
ngram 1=26
ngram 2=31
ngram 3=35

\1-grams:
-99.0000 <UNK>0.0000
-1.5315 AUDITORIO -0.7652
-1.5315 CIUDAD -0.7652
-1.5315 DE -0.7652
-1.5315 EL -0.7652
-1.2884 EN -0.9668
-1.5315 ESCUCHA -0.7552
-1.5315 HAY -0.7760
-1.6564 INFORMACION -0.6893
-1.6564 INFORMATIVO -0.6692
-1.2884 LA -0.9771
-1.6564 MAS -0.6760
-1.5315 MEXICO -0.7652
...
\2-grams:
-0.0792 AUDITORIO QUE 0.0792
-0.0792 CIUDAD DE 0.0792
-0.0792 DE MEXICO 0.0792
-0.0792 EL AUDITORIO 0.0792
-0.3010 EN LA -0.3979
-0.3979 EN UN 0.0969
-0.0792 ESCUCHA EN -0.3979
-0.5441 HAY EN -0.0792
-0.2430 HAY PARA -0.4508
...
\3-grams:
-0.0969 AUDITORIO QUE NOS
-0.0969 CIUDAD DE MEXICO
-0.0969 DE MEXICO HAY
-0.0969 EL AUDITORIO QUE
-0.0969 EN LA CIUDAD
-0.1249 EN UN MOMENTO
-0.0969 ESCUCHA EN LA
-0.3010 HAY EN UN
-0.1249 HAY PARA EL
-0.3010 INFORMACION VIAL EN
-0.6021 INFORMACION VIAL PARA
-0.1249 INFORMATIVO OCHENTA Y
...
\end\
```

El formato ARPA consta de varias secciones: La sección de datos `\data\` muestra el número de n-gramas que se generó en cada sección. En *Sphinx* sólo se soporta hasta modelos de trigramas. Cada sección `\n-gram\` en donde $n = 1, 2,$ o 3. especifica las probabilidades logarítmicas (en log_{10}) y los pesos `Back-off` calculados para los $n-$gramas. La sección `\end\` termina el archivo arpa. Observe que en la sección de `1-grama` se puede ver la secuencia $-99.0000 < UNK > 0.0000$ lo que indica que le asigna una probabilidad de 0.0 a palabras que se encuentran fuera del vocabulario para descartarlas en el reconocimiento.

6

Pruebas y Resultados

El corpus grabado consiste de 5 horas de grabación del noticiero de la radio 88.9 noticias, Panorama informativo en FM. Se descartaron aquellas frases que contienen ruido excesivo y que presentan traslape de voces para tratar de mejorar la precisión del reconocimiento. La base de datos ocupa 360 MB en disco duro. Se trata de 21 hablantes, de los cuales 15 son hombres y 6 son mujeres. Ahora bien, debido a las limitantes de tiempo, sólo escogí trabajar con 2 personas del sexo femenino. La duración de las grabaciones de BEATRIZC es de 5 minutos, 5 segundos y 405 milésimas ($5'05''405$) y se usó para reconocer 4 frases del español que se repetían con mayor frecuencia dentro del corpus de entrenamiento. La cantidad de palabras que posee esta parte es de 25 palabras en total. La duración de las grabaciones de la persona ARELIP es de 1 minuto, 1 segundo y 138 milésimas ($1'01''138$) y que corresponde al entrenamiento de las 109 palabras sin repetición del español hablado en México que se desean reconocer. La razón de tomar 109 palabras en lugar de 100 es que, en realidad, se eligieron 13 oraciones que la persona en cuestión pronunció.
El formato en el que se grabaron los archivos es formato raw PCM de 16 bits, con frecuencia de muestreo es de 16 kHz, un sólo canal, codificación big-endian.

Se han omitido los parámetros de las aplicaciones del entrenamiento, de la decodificación y del modelado del lenguaje de *Sphinx* al igual que muchas de sus corridas debido a que es bastante infomación, sin embargo, para no perder la investigación realizada consúltese la página http://profesores.fi-b.unam.mx/jareyc/procvoz.html con el fin de obtener información más detallada y complementar este trabajo.

6.1. Reconocimiento de frases del español hablado en la región central de México

6.1.1. Creación del archivo de definición del modelo CI

La aplicación `mk_mdef` generó el archivo de definición del modelo para el entrenamiento de los modelos CI, el cuál se muestra a continuación:

```
# Generated by bin\mk_mdef_gen on Sun Aug 30 18:37:17 2009
0.3
22 n_base
0 n_tri
88 n_state_map
66 n_tied_state
66 n_tied_ci_state
22 n_tied_tmat
#
# Columns definitions
#base lft  rt p attrib tmat      ... state id's ...
    A   -   - -    n/a    0    0    1    2    N
  A_1   -   - -    n/a    1    3    4    5    N
    B   -   - -    n/a    2    6    7    8    N
   CH   -   - -    n/a    3    9   10   11    N
    D   -   - -    n/a    4   12   13   14    N
    E   -   - -    n/a    5   15   16   17    N
  E_1   -   - -    n/a    6   18   19   20    N
    F   -   - -    n/a    7   21   22   23    N
    I   -   - -    n/a    8   24   25   26    N
    K   -   - -    n/a    9   27   28   29    N
    L   -   - -    n/a   10   30   31   32    N
    M   -   - -    n/a   11   33   34   35    N
    N   -   - -    n/a   12   36   37   38    N
    O   -   - -    n/a   13   39   40   41    N
  O_1   -   - -    n/a   14   42   43   44    N
    P   -   - -    n/a   15   45   46   47    N
    R   -   - -    n/a   16   48   49   50    N
    S   -   - -    n/a   17   51   52   53    N
  SIL   -   - - filler   18   54   55   56    N
    T   -   - -    n/a   19   57   58   59    N
    U   -   - -    n/a   20   60   61   62    N
    X   -   - -    n/a   21   63   64   65    N
```

Al observar el archivo de definición del modelo se tiene: la primera línea es un comentario que describe el nombre de la herramienta que se utilizó junto con la fecha y la hora de creación del archivo. La siguiente línea indica la versión del software, con el cual fue creado el archivo (para este ejemplo `ver` tiene 0.3). Las siguientes seis líneas definen los parámetros utilizados para la construcción del modelo, En la tabla 6.1 se explica cada uno de ellos. Las líneas restantes son atributos que se observan en la tabla 6.2.

Así, del archivo se observa que se obtuvieron 22 fonemas base (`n_base`), ordenados alfabéticamente y considerando el silencio (`SIL`) como parte de ellos. 0 trifonemas (`n_tri`), puesto que se trata de modelos CI. 88 estados en total, 66 emitidos y 22 no emitidos, de los cuáles los emitidos están numerados del 0 al 65 y los no emitidos tienen el valor `N`. 66 estados ligados (`n_tied_state`), en este caso no se generaron estados ligados, así que son los mismos que el total de estados emitidos. 66 fonemas base después de haberlos compartido (`n_tied_ci_state`), en este caso no se compartieron, así que son los mismos que el total de estados emitidos. 22 matrices de transición (`n_tied_tmat`), numeradas del 0 al 21. Los fonemas base no tienen atributo (n/a), excepto `SIL` que tiene el atributo de relleno (filler). Además carecen de contextos tanto izquierdo (lft) como derecho (rt) y tampoco se encuentran en ningún trifonema (p).

Parámetro	Descripción
n_base	Número de fonemas base o fonemas CI que se tienen.
n_tri	Número de trifonemas
n_state_map	Número total de estados HMM emitidos y no emitidos. Recuerde que para cada HMM *Sphinx* agrega un estado extra de terminación que es no emitido. Por ejemplo, si cada fonema es modelado por HMMs de 3 estados y si se tienen 22 fonemas, el valor de este parámetro se calcula de la siguiente manera: 22 $(fonemas) \times 4\ (estados) = 88$ estados
n_tied_state	Cantidad de todos los fonemas después de haberlos compartido. No se comparten estados en el entrenamiento de los modelos CI. Por lo tanto este número es el mismo que el número total de estados emitidos, $22 \times 3 = 66$
n_tied_ci_state	Número de estados de los fonemas base después de haberlos compartido. En entrenamiento de los modelos CI el número de fonemas base es el mismo que el número de todos fonemas que son modelados.
n_tied_tmat	El HMM para cada fonema CI tiene asociada una matriz de transición de probabilidades. Este parámetro corresponde al número total de matrices de transición para el conjunto de modelos dado.

Tabla 6.1: Parámetros del archivo de definición del modelo

Columna	Descripción
base	Nombre de cada fonema
lft	El contexto de la izquierda del fonema (left-context). Si no existe se pone -.
rt	El contexto de la derecha del fonema (right-context). Si no existe se pone -.
p	Posición del trifonema. Posee cuatro marcadores de posición: b indica que el trifonema está al inicio de la palabra, e indica que el trifonema está al final de la palabra, i indica que el trifonema está en medio de la palabra (word internal triphone) y s indica que se trata un único trifonema . No se requiere en el entrenamiento CI.
attrib	Atributos del fonema. En la lista de fonemas, si el fonema es "SIL", o si el fonema es encerrado por "+", como en "$+BANG+$", *Sphinx* interpreta ese fonema como evento de no-voz (non-speech). También existen fonemas llamados de relleno y su atributo se asigna como "filler". Los fonemas base no tienen atributos especiales y por lo tanto son etiquetados como "n/a", que significa "ningún atributo"
tmat	El identificador único (id) de la matriz de transición, asociada con el fonema.
state_id's	Los identificadores únicos (ids) de los estados asociados HMM de cualquier fonema. Esta lista es terminada por una "N" que se entiende como un estado no-emitido. Ningún id es asociado a este último estado. Sin embargo, éste existe y es listado.

Tabla 6.2: Parámetros del archivo de definición del modelo

6.1.2. Creación del archivo de topología HMM

El contenido del archivo de toplogía ya fue explicado en la sección 5.1.2 del capítulo 5 por lo que no se repetirá.

6.1.3. Inicialización plana de los parámetros CI

Es importante señalar que todos los archivos que hayan sido creados con las aplicaciones de *Sphinx* llevan un *encabezado*, que tiene la siguiente forma:

```
s3
version 1.0
chksum0 yes
endhdr
```

El primer renglón del encabezado indica que se trata de un archivo creado por *Sphinx 3* . La siguiente línea contiene la versión de la aplicación que manipuló el archivo y que varía de acuerdo al software que se este empleando, en el ejemplo la versión es 1.0. La tercera línea indica si se verifica la información por un checksum[1] o no se verifica. Y por último, la cuarta línea muestra el indicador de fin de encabezado.

Inicialización de pesos mixtos y matrices de transición

La aplicación `mk_flat` genera las matrices de transición con el siguiente formato:

```
tmat 22 4
tmat [0]
 5.000e-001 5.000e-001
          5.000e-001 5.000e-001
                   5.000e-001 5.000e-001
tmat [1]
 5.000e-001 5.000e-001
          5.000e-001 5.000e-001
                   5.000e-001 5.000e-001
.
.
.
```

En la primera línea, `tmat` indica el nombre del arreglo que contiene a las matrices de transición. Los valores que le siguen a `tmat` corresponden a las dimensiones del arreglo. La primera dimensión, en este caso el número 22, indica el número de fonemas independientes del contexto que existen en el diccionario de unidades fonéticas y que también corresponde a la cantidad de matrices de transición que fueron creadas. La segunda dimensión, en este caso el número 4, indica la cantidad de estados, tanto emitidos como no emitidos, que tiene cada matriz y que corresponde al número de estados definidos en el modelo de topología. En los renglones siguientes se presenta cada matriz de transición definida por el nombre del arreglo y su respectiva posición a la que hace referencia el índice entre corchetes. Observe también que sólo se guardan aquellos valores cuyas probabilidades de transición son distintas de

[1] checksum es un dato que comprueba, mediante un algoritmo, si los datos almacenados son correctos.El algoritmo que se utiliza varia dependiendo del archivo y de la aplicación que se manejen.

cero colocadas en las respectivas posiciones de sus estados de acuerdo a como se definieron en el archivo de topología.

Por su parte, la aplicación `mk_flat` almacena los pesos mixtos, de la siguiente manera:

```
mixw 66 1 1
mixw [0 0] 1.000000e+000

1.000e+000
mixw [1 0] 1.000000e+000

1.000e+000
.
.
.
```

En forma similar al archivo de matrices de transición `mixw` indica el nombre del arreglo que contiene a los pesos mixtos; le siguen a éste las dimensiones del arreglo. La primera dimensión, el número 66, corresponde a la cantidad de estados emitidos que se generaron. La segunda dimensión, es decir, el primer 1, corresponde al número de flujos de características que se manejan por estado; y la tercera dimensión, es decir, el segundo 1, corresponde al número de densidades que se manejan por cada estado. En los siguientes renglones se presenta cada matriz de pesos mixtos haciendo referencia a ellos mediante el nombre del arreglo y de sus respectivas posiciones dadas de acuerdo al estado emitido y al número de flujos de características.

Inicialización de la media y la variancia globales

Como resultado de haber ejecutado las aplicaciones `init_gau` y `norm` se obtuvo la media global, la cuál se almacenó a manera de un arreglo de dimensiones $[1 \times 1 \times 1]$ en un archivo cuyo formato presenta lo siguiente:

```
param 1 1 1
mgau 0
feat 0
density    0 1.917e-008 9.074e-009 -1.054e-009 3.727e-009 -1.225e-009 2.068e-009 ...
```

En la primera línea, la etiqueta `param` indica que se trata de un archivo que puede contener a un arreglo de vectores de medias o a un arreglo de matrices de covariancia. Los siguientes tres valores en esa línea se asocian con las dimensiones del arreglo, según corresponda: La primera dimensión, el primer 1, corresponde a la cantidad de estados emitidos que contienen a esa media o a esa variancia. La segunda dimensión, el segundo 1, corresponde al número de flujos de características utilizados y la tercera dimensión, el tercer 1, indica la cantidad de densidades Gaussianas por estado que tiene la media. Así, en este ejemplo se indica que hay 1 estado emitido con 1 flujo de características y 1 densidad Gaussiana, debido a que se trata de la media global.
El parámetro `mgau` indica el identificador del estado emitido al que pertenece la media, en este caso, como solamente hay uno, su identificador es 0. El parámetro

`feat` indica el identificador del flujo de características al que pertenece esa media, en forma similar, como solamente se maneja un flujo de características en modelos HMM continuos, su ídentificador toma el valor de 0. El parámetro `density` junto con el valor que le sigue (0) señalan el identificador de la densidad y los demás valores corresponden a los coeficientes del vector de la media global, que en este caso como son 39 solamente se muestran los 6 primeros coeficientes.

Como resultado de ejecutar las aplicaciones `init_gau` y `norm`, de nueva cuenta, se obtuvo la variancia global, que también se almacenó como un arreglo de dimensiones $[1 \times 1 \times 1]$ en su respectivo archivo y que presenta un formato como se muestra a continuación:

```
param 1 1 1
mgau 0
feat 0
density    0 4.214e+000 3.152e-001 1.611e-001 1.339e-001 8.691e-002 6.367e-002...
```

Como se observa, es el mismo formato que el descrito para la media; la diferencia radica en que la asociación de parámetros se realiza con respecto a la variancia global obtenida y los valores de los coeficientes corresponden a los valores de su matriz diagonal, que se configuró para simplificar cálculos.

Copia de la media y variancia globales a cada estado HMM

Como resultado de la copia de la media global a cada estado HMM , mediante la aplicación `cp_parm`, se obtuvo un archivo como el siguiente:

```
param 66 1 1
mgau 0
feat 0
density    0 1.917e-008 9.074e-009 -1.054e-009 3.727e-009 -1.225e-009 2.068e-009 ...
mgau 1
feat 0
density    0 1.917e-008 9.074e-009 -1.054e-009 3.727e-009 -1.225e-009 2.068e-009 ...
mgau 2
feat 0
density    0 1.917e-008 9.074e-009 -1.054e-009 3.727e-009 -1.225e-009 2.068e-009 ...
...
mgau 65
feat 0
density    0 1.917e-008 9.074e-009 -1.054e-009 3.727e-009 -1.225e-009 2.068e-009 ...
```

Aquí se puede observar lo que se comentó con respecto al formato del archivo de la media global, es decir, ahora la primer dimensión del arreglo, en este caso, es de 66, por lo que se tiene la misma cantidad de vectores de medias asociados con su respectivo identificador de su estado emitido mediante el parámetro `mgau`. Se sigue manejando un solo vector de características y una densidad por cada estado. Observe también que los coeficientes de cada vector corresponden a los valores de la media global.

Una situación similar se presenta con los resultados de la copia de la variancia global a cada estado HMM, en donde ahora se observan 66 matrices de covariancia diagonales asociadas a su respectivo estado HMM emitido, como se observa a continuación:

```
param 66 1 1
mgau 0
feat 0
```

```
density    0 4.214e+000 3.152e-001 1.611e-001 1.339e-001 8.691e-002 6.367e-002...
mgau 1
feat 0
density    0 4.214e+000 3.152e-001 1.611e-001 1.339e-001 8.691e-002 6.367e-002...
mgau 2
feat 0
density    0 4.214e+000 3.152e-001 1.611e-001 1.339e-001 8.691e-002 6.367e-002...
...
mgau 65
feat 0
density    0 4.214e+000 3.152e-001 1.611e-001 1.339e-001 8.691e-002 6.367e-002...
```

6.1.4. Entrenamiento de los modelos independientes del contexto (CI)

Lo que cabe destacar de la ejecución de la aplicación `bw` son los resultados correspondientes a las estadísticas que se obtienen para cada pronunciación, las que se presentan como se muestra a continuación:

```
...
column defns
<seq>
<id>
<n_frame_in>
<n_frame_del>
<n_state_shmm>
<avg_states_alpha>
<avg_states_beta>
<avg_states_reest>
<avg_posterior_prune>
<frame_log_lik>
<utt_log_lik>
... timing info ...
...
utt>     0        BC01F  226    0   144 109  34 69 4.356861e-011 -9.315862e+000 -2.105385e+003
utt>     1        BC02F  222    0   144 109  34 68 4.606641e-011 -9.152735e+000 -2.031907e+003
utt>     2        BC03F  206    0   144 106  33 65 3.909968e-011 -1.532036e+001 -3.155994e+003
utt>     3        BC04F  239    0   144 111  35 71 4.657089e-011 -9.923626e+000 -2.371747e+003
utt>     4        BC05F  229    0   144 110  34 69 4.251009e-011 -8.556417e+000 -1.959419e+003
utt>     5        BC06F  216    0   144 108  33 67 4.037081e-011 -1.011888e+001 -2.185679e+003
utt>     6        BC01F  344    0   200 156  43 86 5.256954e-011 -1.039372e+001 -3.575441e+003
utt>     7        BC02F  358    0   200 157  44 87 5.305952e-011 -1.119092e+001 -4.006349e+003
utt>     8        BC03F  319    0   200 153  41 83 5.025075e-011 -9.690696e+000 -3.091332e+003
utt>     9        BC04F  301    0   200 150  40 81 5.043904e-011 -1.233505e+001 -3.712850e+003
utt>    10        BC01F  263    0   140 112  36 73 4.449036e-011 -9.798483e+000 -2.577001e+003
utt>    11        BC02F  236    0   140 109  35 70 4.459888e-011 -1.140277e+001 -2.691053e+003
utt>    12        BC03F  270    0   140 112  37 74 4.915557e-011 -9.180129e+000 -2.478635e+003
utt>    13        BC01F  365    0   220 169  44 89 5.411214e-011 -1.283753e+001 -4.685698e+003
utt>    14        BC02F  371    0   220 169  45 90 5.326253e-011 -1.074993e+001 -3.988224e+003
utt>    15        BC03F  361    0   220 168  44 89 5.537869e-011 -1.383713e+001 -4.995205e+003
overall> WIN32(N/A) 4526 (-0) -1.096154e+001 -4.961192e+004
...
```

Observe que `bw` genera una tabla con las pronunciaciones. El significado de cada columna se lista en la tabla 6.3 en el orden en que aparecen durante la ejecución de la aplicación.

Una vez que se han terminado de procesar todas las pronunciaciones, la etiqueta `overall>`indica un resumen final con el nombre de la maquina (WIN32 N/A), que en este caso no está disponible, el número total de tramas (4526), el número total de tramas eliminadas (-0), la probabilidad logarítmica por trama de todas las pronunciaciones ($-1.096154e+001$), la probabilidad logarítmica

Notación	Descripción
< `seq` >	Número de pronunciación procesada.
< `id` >	Identificador de la pronunciación en el archivo de control.
< `n_frame_in` >	Numero de tramas por pronunciación.
< `n_frame_del` >	Numero de tramas eliminadas.
< `n_state_shmm` >	Número de estados.
< `avg_states_alpha` >	Promedio de estados activos usados en el algoritmo hacia adelante (forward).
< `avg_states_beta` >	Promedio de estados activos en el procedimiento hacia atrás (backward).
< `avg_states_reest` >	Promedio de estados utilizados en la reestimación.
< `avg_posterior_prune` >	Promedio de la probabilidad a posteriori recortada.
< `frame_log_lik` >	Probabilidad logarítmica por trama.
< `utt_log_lik` >	Probabilidad logarítmica de la pronunciación.
...`timing info`...	Tiempo consumido durante el proceso de estimación de los parámetros.

Tabla 6.3: Descripción de la salidas de la aplicación `bw`.

total de las pronunciaciones dado el modelo actual ($-4.961192e+004$) y el tiempo transcurrido en el proceso de estimación no se muestra.

Recuerde que los valores obtenidos para los parámetros de los modelos HMM (medias, variancias, matrices de transición y pesos mixtos) de las pronunciaciones son solamente los valores acumulados de dichos parámetros, por lo que fue necesario normalizarlos. Así, por medio de la aplicación `norm` se llevó a cabo este proceso y se obtuvieron los respectivos archivos de parámetros del modelo CI.

Sin embargo, recuerde también que al completar las ejecuciones de las aplicaciones `bw` y `norm` sólo se ha realizado el proceso de entrenamiento para una sola iteración, por lo que se aplicó el mismo procedimiento durante 7 iteraciones más. En la tabla 6.4 se muestran las probabilidades logarítmicas por trama obtenidas en cada iteración, así como los criterios de convergencia para continuar con las iteraciones, obtenidos a partir de la segunda iteración. La probabilidad obtenida en la última iteración (8) corresponde a la probabilidad final por trama con la que se finalizó el entrenamiento CI.

Iteración	Probabilidad total por trama	Radio de convergencia
1	−10.9615377817057	–
2	−8.25868979231109	0.246575621342613
3	−4.91836500220946	0.404461830399721
4	−2.81496022978347	0.427663414870812
5	−2.18467012814848	0.223907284716231
6	−1.99585859478568	0.0864256488565695
7	−1.91138996906761	0.0423219490292302
8	−1.84273641184269	0.0359181320065276

Tabla 6.4: Criterios de convergencia para el entrenamiento de modelos CI.

Los archivos de los parámetros del modelo CI poseen contenidos similares a los que se describieron en la sección 6.1.3 por lo que no se incluirán en este trabajo, lo que cambió fueron los valores obtenidos durante la reestimación pero su formato se mantuvo.

6.1.5. Creación del archivo de definición del modelo no ligado dependiente del contexto

De acuerdo a como se especificó en la sección 5.1.5 se generaron dos archivos de definición de modelos CD no ligados, un *archivo de definición del modelo para todos los trifonemas que ocurren en el cuerpo de entrenamiento* y un *archivo de definición del modelo que incluye solamente una lista corta de trifonemas y fonemas CI.*

Un segmento del archivo de definición del modelo que lista todos los posibles trifonemas se muestra a continuación:

```
# Generated by C:\TESISMAESTRIA\Sphinx\Tesis\bin\mk_mdef_gen.exe on Fri Oct 23 20:29:40 2009
0.3
22 n_base
692 n_tri
2856 n_state_map
2142 n_tied_state
66 n_tied_ci_state
22 n_tied_tmat
#
# Columns definitions
#base lft  rt p attrib tmat      ... state id's ...
    A   -   - -    n/a    0    0    1    2    N
  A_1   -   - -    n/a    1    3    4    5    N
    B   -   - -    n/a    2    6    7    8    N
   CH   -   - -    n/a    3    9   10   11    N
    D   -   - -    n/a    4   12   13   14    N
    E   -   - -    n/a    5   15   16   17    N
  E_1   -   - -    n/a    6   18   19   20    N
    F   -   - -    n/a    7   21   22   23    N
    ...
    R   -   - -    n/a   16   48   49   50    N
    S   -   - -    n/a   17   51   52   53    N
  SIL   -   - - filler   18   54   55   56    N
    T   -   - -    n/a   19   57   58   59    N
    U   -   - -    n/a   20   60   61   62    N
    X   -   - -    n/a   21   63   64   65    N
    A   A   I b    n/a    0   66   67   68    N
    A   A   U b    n/a    0   69   70   71    N
    A  CH   A e    n/a    0   72   73   74    N
    A  CH   B e    n/a    0   75   76   77    N
    A  CH   D e    n/a    0   78   79   80    N
    A  CH   E e    n/a    0   81   82   83    N
    A  CH   I e    n/a    0   84   85   86    N
    A  CH   K e    n/a    0   87   88   89    N
    A  CH   L e    n/a    0   90   91   92    N
    A  CH   M e    n/a    0   93   94   95    N
    A  CH   N e    n/a    0   96   97   98    N
    A  CH   O e    n/a    0   99  100  101    N
    ...
    S   O   M e    n/a   17 2055 2056 2057    N
    S   O   N e    n/a   17 2058 2059 2060    N
    S   O   O e    n/a   17 2061 2062 2063    N
    S   O   P e    n/a   17 2064 2065 2066    N
    S   O   S e    n/a   17 2067 2068 2069    N
    S   O SIL e    n/a   17 2070 2071 2072    N
```

```
S   O   U e    n/a   17 2073 2074 2075    N
S   S   I b    n/a   17 2076 2077 2078    N
S SIL   I b    n/a   17 2079 2080 2081    N
T   A   I i    n/a   19 2082 2083 2084    N
T   I   O i    n/a   19 2085 2086 2087    N
T   N   A i    n/a   19 2088 2089 2090    N
T   N   O i    n/a   19 2091 2092 2093    N
T   O   I i    n/a   19 2094 2095 2096    N
U   A   D i    n/a   20 2097 2098 2099    N
U   A   N b    n/a   20 2100 2101 2102    N
U   D   N b    n/a   20 2103 2104 2105    N
U   E   N b    n/a   20 2106 2107 2108    N
U   I   D i    n/a   20 2109 2110 2111    N
U   I   N b    n/a   20 2112 2113 2114    N
U   K  CH i    n/a   20 2115 2116 2117    N
U   L   N b    n/a   20 2118 2119 2120    N
U   N   E i    n/a   20 2121 2122 2123    N
U   N   N b    n/a   20 2124 2125 2126    N
U   O   N b    n/a   20 2127 2128 2129    N
U   P   N i    n/a   20 2130 2131 2132    N
U   S   N b    n/a   20 2133 2134 2135    N
U SIL   N b    n/a   20 2136 2137 2138    N
X E_1   I i    n/a   21 2139 2140 2141    N
```

Observe que su contenido es similar al del archivo de definición del modelo CI, visto en la sección 6.1.1. Incluye los modelos de los fonemas base y se distingue del anterior porque ahora se incluyen también todos los trifonemas obtenidos, con sus respectivos contextos derecho e izquierdo y la respectiva posición que ocupan dentro de la palabra, tal y como se especificó en las tablas 6.1 y 6.2.

En este archivo se observa que, además de los 22 fonemas base, se obtuvieron los 692 trifonemas, 2856 estados en total, de los cuales 2142 son emitidos y 714 no emitidos; los identificadores de los estados emitidos están numerados del 0 al 2141 y los de los no emitidos solamente tienen el identificador `N`. Se tienen también 2142 estados ligados (`n_tied_state`), debido a que todavía no se comparten. 66 fonemas base después de haberlos compartido (`n_tied_ci_state`) y 22 matrices de transición (`n_tied_tmat`), cuyos identificadores están numeradas del 0 al 21.

Por otra parte, para el *archivo de definición del modelo no ligado CD que incluye solamente una lista corta de trifonemas y fonemas CI* se obtuvieron, además de los 22 fonemas base, 118 trifonemas, 560 estados en total, de los cuáles 420 son emitidos y 140 son no emitidos; los identificadores de los emitidos están numerados del 0 al 419 y los de los no emitidos tienen el valor `N`. Hay también 420 estados ligados (`n_tied_state`), 66 fonemas base después de haberlos compartido (`n_tied_ci_state`) y 22 matrices de transición (`n_tied_tmat`), numeradas del 0 al 21.

El archivo de definición del modelo no ligado CD que incluye solamente una lista corta de trifonemas y fonemas CI no se muestra debido a que su contenido es bastante similar al archivo de definición del modelo no ligado CD que lista todos los posibles trifonemas.

6.1.6. Inicialización plana de los parámetros no ligados dependientes del contexto

Lo que cabe destacar de la ejecución de la aplicación `init_mixw`, que se encarga de copiar los parámetros CI a los parámetros no ligados CD, es lo siguiente:

```
        A <- A          : [tm     0 +=     0]
[mx     0(0) +=     0] [mg     0(0) <-     0] [mx     1(1) +=     1] [mg     1(1) <-     1]
 [mx     2(2) +=     2] [mg     2(2) <-     2]
       A_1 <- A_1        : [tm     1 +=     1]
[mx     3(0) +=     3] [mg     3(0) <-     3] [mx     4(1) +=     4] [mg     4(1) <-     4]
 [mx     5(2) +=     5] [mg     5(2) <-     5]
        B <- B          : [tm     2 +=     2]
[mx     6(0) +=     6] [mg     6(0) <-     6] [mx     7(1) +=     7] [mg     7(1) <-     7]
 [mx     8(2) +=     8] [mg     8(2) <-     8]
       CH <- CH         : [tm     3 +=     3]
...
      SIL <- SIL        : [tm    18 +=    18]
[mx    54(0) +=    54] [mg    54(0) <-    54] [mx    55(1) +=    55] [mg    55(1) <-    55]
 [mx    56(2) +=    56] [mg    56(2) <-    56]
        T <- T          : [tm    19 +=    19]
[mx    57(0) +=    57] [mg    57(0) <-    57] [mx    58(1) +=    58] [mg    58(1) <-    58]
 [mx    59(2) +=    59] [mg    59(2) <-    59]
        U <- U          : [tm    20 +=    20]
[mx    60(0) +=    60] [mg    60(0) <-    60] [mx    61(1) +=    61] [mg    61(1) <-    61]
 [mx    62(2) +=    62] [mg    62(2) <-    62]
        X <- X          : [tm    21 +=    21]
[mx    63(0) +=    63] [mg    63(0) <-    63] [mx    64(1) +=    64] [mg    64(1) <-    64]
 [mx    65(2) +=    65] [mg    65(2) <-    65]
  A CH E e <- A         : [mx    66(0) +=     0] [mg    66(0) <-     0] [mx    67(1) +=     1]
 [mg    67(1) <-     1] [mx    68(2) +=     2] [mg    68(2) <-     2]
   A D D i <- A         : [mx    69(0) +=     0] [mg    69(0) <-     0] [mx    70(1) +=     1]
 [mg    70(1) <-     1] [mx    71(2) +=     2] [mg    71(2) <-     2]
   A I L i <- A         : [mx    72(0) +=     0] [mg    72(0) <-     0] [mx    73(1) +=     1]
 [mg    73(1) <-     1] [mx    74(2) +=     2] [mg    74(2) <-     2]
   A I S i <- A         : [mx    75(0) +=     0] [mg    75(0) <-     0] [mx    76(1) +=     1]
 [mg    76(1) <-     1] [mx    77(2) +=     2] [mg    77(2) <-     2]
   A L I e <- A         : [mx    78(0) +=     0] [mg    78(0) <-     0] [mx    79(1) +=     1]
 [mg    79(1) <-     1] [mx    80(2) +=     2] [mg    80(2) <-     2]
   A L S e <- A         : [mx    81(0) +=     0] [mg    81(0) <-     0] [mx    82(1) +=     1]
...
   T O I i <- T         : [mx   396(0) +=    57] [mg   396(0) <-    57] [mx   397(1) +=    58]
 [mg   397(1) <-    58] [mx   398(2) +=    59] [mg   398(2) <-    59]
   U A D i <- U         : [mx   399(0) +=    60] [mg   399(0) <-    60] [mx   400(1) +=    61]
 [mg   400(1) <-    61] [mx   401(2) +=    62] [mg   401(2) <-    62]
   U I D i <- U         : [mx   402(0) +=    60] [mg   402(0) <-    60] [mx   403(1) +=    61]
 [mg   403(1) <-    61] [mx   404(2) +=    62] [mg   404(2) <-    62]
  U K CH i <- U         : [mx   405(0) +=    60] [mg   405(0) <-    60] [mx   406(1) +=    61]
 [mg   406(1) <-    61] [mx   407(2) +=    62] [mg   407(2) <-    62]
   U N E i <- U         : [mx   408(0) +=    60] [mg   408(0) <-    60] [mx   409(1) +=    61]
 [mg   409(1) <-    61] [mx   410(2) +=    62] [mg   410(2) <-    62]
   U N N b <- U         : [mx   411(0) +=    60] [mg   411(0) <-    60] [mx   412(1) +=    61]
 [mg   412(1) <-    61] [mx   413(2) +=    62] [mg   413(2) <-    62]
   U P N i <- U         : [mx   414(0) +=    60] [mg   414(0) <-    60] [mx   415(1) +=    61]
 [mg   415(1) <-    61] [mx   416(2) +=    62] [mg   416(2) <-    62]
 X E_1 I i <- X         : [mx   417(0) +=    63] [mg   417(0) <-    63] [mx   418(1) +=    64]
 [mg   418(1) <-    64] [mx   419(2) +=    65] [mg   419(2) <-    65]
```

De acuerdo a como se comentó en la sección 5.1.6, se realizan las copias de los parámetros desde los modelos CI a los modelos CD. En primer lugar para los fonemas 22 fonemas base y luego para los trifonemas. Observe que los fonemas base CI se copian para cada estado de cada fonema no ligado CD de la siguiente

manera:

```
fonema_baseno_ligado_CD ← fonema_base_CI :
[matriz_de_transición ID_estado_no_ligado_CD + = ID_estado_CI]
[peso_mixto ID_estado_no_ligado_CD(estado_actual) + = ID_estado_CI]
[media_y_variancia ID_estado_no_ligado_CD(estado_actual) + = ID_estado_CI]
```
(6.1)

En forma similar los trifonemas se copian para cada estado no ligado CD a partir de los fonemas base CI de la siguiente manera:

```
trifonema_no_ligado_CD ← fonema_base_CI :
[peso_mixto ID_estado_no_ligado_CD(estado_actual) + = ID_estado_CI]
[media_y_variancia ID_estado_no_ligado_CD(estado_actual) + = ID_estado_CI]
```
(6.2)

donde `matriz_de_transición` se representa como `tm`, `peso_mixto` se representa como `mx`, `media_y_variancia` se representan como `mg`, `ID_estado_no_ligado_CD` corresponde al identificador del estado no ligado CD de destino, `ID_estado_CI` corresponde al identificador del estado CI de origen y `estado_actual` corresponde a cada uno de los valores de los estados HMM que definen ya sea al fonema base o al fonema CI y cuyos valores son, en este caso, $0, 1$ o 2.

6.1.7. Entrenamiento de los modelos no ligados CD

Como el entrenamiento de los modelos no ligados CD se hace iterando varias veces las aplicaciones `bw` y `norm` hasta que se alcance el criterio de convergencia establecido y que las salidas de las corridas para estas aplicaciones son muy similares a las tratadas en la sección 6.1.4 no se mostrarán dichas corridas. En cambio, en la tabla 6.5 se mostrarán las probabilidades obtenidas por trama y el radio de convergencia obtenidos en cada iteración. Observe que en este caso la convergencia se obtuvo en la cuarta iteración.

Iteración	Probabilidad total por trama	Radio de convergencia
1	−1.77581131241715	–
2	6.46877817057004	4.64271706421616
3	10.775	0.665693229213097
4	10.9770326999558	0.0187501345666646

Tabla 6.5: Criterios de convergencia para el entrenamiento de modelos no ligados CD.

6.1.8. Construcción de los árboles de decisión para compartir parámetros

Generación de las preguntas lingüísticas

En las siguientes líneas se muestra un segmento del contenido del archivo de preguntas lingüísticas construido para el entrenamiento de las 4 frases del español y que fue obtenido por medio del ejecutable `make_quests`.

```
WDBNDRY_B
WDBNDRY_E
```

```
WDBNDRY_S
WDBNDRY_I
SILENCE   SIL
QUESTION0_0_R  A A_1 B E I L M O O_1 R U X
QUESTION0_1_R  A A_1 B L R U
QUESTION0_2_R  A
QUESTION0_3_R  A_1 B L R U
QUESTION0_4_R  A_1 B L U
QUESTION0_5_R  A_1 B U
QUESTION0_6_R  B U
...
QUESTION0  E E_1 F O O_1 S T
QUESTION1  S
QUESTION2  E E_1 F O O_1 T
QUESTION3  E F O
QUESTION4  F O
...
QUESTION2_0_L  E O S
QUESTION2_1_L  S
QUESTION2_2_L  E O
QUESTION2_3_L  A A_1 B CH D E_1 F I K L M N O_1 P R T U X
QUESTION2_4_L  E_1 F L N O_1 P R T X
...
```

Observe que en los primeros cuatro renglones se almacenan las preguntas correspondientes a la posición del trifonema dentro de una palabra: la etiqueta `WDBNDRY_B` indica si éstos se encuentran al inicio de una palabra, la etiqueta `WDBNDRY_E` indica si éstos se encuentran al final de una palabra, la etiqueta `WDBNDRY_S` indica si el trifonema es único y la etiqueta `WDBNDRY_I` indica si el trifonema se encuentra dentro de la palabra.
Le siguen las preguntas lingüísticas que tienen la forma `QUESTIONn_m_P`, donde `n` representa la posición del estado dentro del modelo acústico; `m` representa el número de la pregunta; el valor de `P` indica la ubicación de `n` dentro de los estados del modelo y puede tomar alguno de los siguientes valores: `R` cuando `n` se encuentra en la primera mitad de los estados dentro del modelo; `L` cuando `n` se encuentra en la segunda mitad de los estados dentro del modelo acústico; cuando se sitúa exactamente en medio no se pone ninguna etiqueta. Aunada a cada pregunta se encuentra una lista ordenada de los fonemas existentes en ese nodo.

Construcción de los árboles de decisión

Lo que se puede destacar de la corrida de la aplicación `bldtree`, que se uso para crear los árboles de decisión para cada estado de cada fonema, es lo siguiente:

```
s>    ( QUESTION15 -1 -4.482e+002 6.356e+002 1.550e+002
s>        ( QUESTION0_15_R 1 9.799e+002 3.404e+002 5.325e+001
s>            ( - 4.298e+002 1 0)
s>            ( - 8.906e+002 2 0 QUESTION0_0_R 1 2.520e+002 4.124e+001))
s>        ( QUESTION5 -1 -7.926e+002 5.135e+002 1.018e+002
s>            ( - 3.741e+002 1 0)
s>            ( QUESTION0_4_R 1 -6.532e+002 4.331e+002 8.976e+001
s>                ( - 2.696e+002 1 0)
```

```
s>               ( QUESTION11 1 -4.897e+002 4.129e+002 8.376e+001
s>                   ( - 2.227e+002 2 0 QUESTION0_0_R 1 1.779e+002 2.079e+001)
s>                   ( QUESTION0_16_R 1 -2.995e+002 3.670e+002 6.297e+001
s>                       ( - 2.948e+002 1 0)
s>                       ( QUESTION0_14_R 1 -2.273e+002 2.826e+002 5.219e+001
s>                           ( - 2.182e+002 2 0 WDBNDRY_E 0 1.358e+002 7.000e+000)
s>                           ( - -1.629e+002 8 0 QUESTION0_3_R 1 2.487e+002 4.519e+001)))))))
...
s>    ( QUESTION0_4_R 1 -6.532e+002 4.331e+002 8.976e+001
s>       ( - 2.696e+002 1 0)
s>       ( QUESTION11 1 -4.897e+002 4.129e+002 8.376e+001
s>           ( - 2.227e+002 2 0 QUESTION0_0_R 1 1.779e+002 2.079e+001)
s>           ( QUESTION0_16_R 1 -2.995e+002 3.670e+002 6.297e+001
s>               ( - 2.948e+002 1 0)
s>               ( QUESTION0_14_R 1 -2.273e+002 2.826e+002 5.219e+001
s>                   ( - 2.182e+002 2 0 WDBNDRY_E 0 1.358e+002 7.000e+000)
s>                   ( QUESTION0_3_R 1 -1.629e+002 2.487e+002 4.519e+001
s>                       ( - 2.851e+002 2 0 QUESTION8 -1 1.892e+002 1.919e+001)
s>                       ( QUESTION0_0_R 1 -1.993e+002 2.190e+002 2.600e+001
s>                           ( QUESTION0 -1 -1.209e+002 2.523e+002 2.300e+001
s>                               ( - 1.094e+002 1 0)
s>                               ( - 2.195e+001 4 0 WDBNDRY_E 0 2.464e+002 1.900e+001))
s>                           ( - 1.406e+002 1 0)))))))
...
s>    ( WDBNDRY_B 0 1.466e+002 2.010e+002 2.119e+001
s>       ( - 1.273e+002 1 0)
s>       ( WDBNDRY_E 0 2.203e+002 1.489e+002 1.719e+001
s>           ( - 2.131e+002 1 0)
s>           ( - 1.561e+002 1 0)))
...
s>    ( QUESTION0_2_R 1 1.548e+002 2.183e+002 1.200e+001
s>       ( - 1.119e+002 1 0)
s>       ( - 2.612e+002 1 0))
...
s>    ( WDBNDRY_E 0 2.203e+002 1.489e+002 1.719e+001
s>       ( - 2.131e+002 1 0)
s>       ( - 1.561e+002 1 0))
...
s>    ( WDBNDRY_E 0 2.182e+002 1.358e+002 7.000e+000
s>       ( - 2.031e+002 1 0)
s>       ( - 1.509e+002 1 0))
```

Observe que se construyeron árboles simples y árboles compuestos. Cada nodo de cada árbol se delimita por medio de paréntesis abierto, "(", y paréntesis cerrado, ")", y que se establecen varios niveles para colocar los nodos. Aquellos nodos que tienen la misma indentación pertenecen al mismo nivel. Los hijos de un nodo tienen un nivel de indentación mayor y se encuentran dentro de los paréntesis del nodo padre. Así, el nodo raíz contiene a todos los demás nodos. Cada nodo consta de un nombre, el cuál puede ser el nombre de una pregunta lingüística (`QUESTIONn_m_P`) o un límite de palabra (`WDBNDRY_B`, `WDBNDRY_E`,etc.), que se obtuvieron a partir del cálculo de la mejor división del nodo raíz. Si el nombre está precedido por un signo de admiración "!", la pregunta es interpretada como negada.Si en lugar del nombre aparece el símbolo "-" significa que ese nodo representa un nodo hoja. El valor que le sigue al nombre indica el contexto: Un valor de -1 significa que existe un fonema a la izquierda, un valor de 1 indica que existe un fonema a la derecha y valor de 0 indica que se trata de un límite de palabra.
Los valores que le siguen al contexto, dependiendo de si se trata del nodo hoja o no lo es, indican: Para los nodos que no son hoja, la probabilidad asociada con

ese nodo, el incremento en la probabilidad después de que el nodo en cuestión ha sido dividido usando la pregunta asociada y el conteo de observaciones asociado con el nodo, respectivamente. Y para los nodos que son nodos hoja: se imprime la probabilidad asociada de la hoja, el número de trifonemas representados, seguidos del grupo al que pertenece la pregunta (éste es usado para formar las preguntas compuestas) y cuyo valor puede ser 0 ó 1. Por último, si el nodo hoja está asociado con una pregunta, de ésta se imprime su correpondiente nombre, contexto, probabilidad asociada con ese nodo y el incremento en la probabilidad después de que el nodo en cuestión ha sido dividido usando dicha pregunta.

Como resultado de la ejecución del programa `bldtree`, se crea un archivo con la estructura de árbol de decisión para el fonema y el estado dado. En este caso, se creo el árbol de decisiones para el fonema `A` y el estado 0 (primer estado, dentro del modelo). En el siguiente listado se muestra un segmento del contenido de dicho archivo.

```
n_node 35
0 1 2 6.664020e+002 1.550101e+002 ((!QUESTION15 -1 !QUESTION5 -1))
1 5 6 4.796702e+002 8.976432e+001 ((!QUESTIONO_4_R 1 QUESTION11 1)(QUESTIONO_14_R 1))
5 15 16 2.674786e+002 2.779237e+001 ((!QUESTIONO_0_R 1))
15 29 30 1.533512e+002 1.200000e+001 ((QUESTIONO_14_R 1))
29 33 34 1.358023e+002 7.000000e+000 ((!WDBNDRY_E 0))
33 - - 1.508738e+002 3.000000e+000
34 - - 2.031464e+002 4.000000e+000
30 - - 1.542161e+002 5.000000e+000
16 - - 2.463740e+002 1.579237e+001
6 7 8 4.226478e+002 6.197195e+001 ((!QUESTIONO_4_R 1))
7 9 10 3.503697e+002 5.597195e+001 ((!QUESTIONO_16_R 1))
9 13 14 3.053944e+002 4.519361e+001 ((QUESTIONO_3_R 1)(!QUESTIONO_3_R 1 QUESTIONO_0_R 1 !QUESTIONO -1))
13 19 20 2.511930e+002 3.819361e+001 ((QUESTION15 1 !QUESTION19 -1))
19 - - 2.914178e+002 5.000000e+000
20 21 22 2.444714e+002 3.319361e+001 ((WDBNDRY_E 0 !QUESTIONO_2_R 1 !QUESTION8 -1)(!WDBNDRY_E 0))
21 27 28 2.009593e+002 2.119355e+001 ((!WDBNDRY_B 0))
27 31 32 1.488990e+002 1.719355e+001 ((!WDBNDRY_E 0))
31 - - 1.560736e+002 7.000000e+000
32 - - 2.131317e+002 1.019355e+001
28 - - 1.273005e+002 4.000000e+000
22 23 24 2.182977e+002 1.200006e+001 ((!QUESTIONO_2_R 1))
23 - - 2.611812e+002 8.999983e+000
24 - - 1.119188e+002 3.000075e+000
14 25 26 2.048183e+002 7.000000e+000 ((!WDBNDRY_B 0))
25 - - 1.406173e+002 3.000000e+000
26 - - 1.094495e+002 4.000000e+000
10 - - 2.948176e+002 1.077834e+001
8 - - 2.695907e+002 6.000000e+000
2 3 4 4.826471e+002 6.524575e+001 ((!WDBNDRY_E 0))
3 11 12 3.404296e+002 5.324830e+001 ((!QUESTIONO_15_R 1))
11 17 18 2.519994e+002 4.124483e+001 ((!QUESTIONO_0_R 1))
17 - - 1.062489e+003 3.824483e+001
18 - - 8.006883e+001 3.000000e+000
12 - - 4.298203e+002 1.200347e+001
4 - - 3.741414e+002 1.199745e+001
```

La información que tiene el archivo es la siguiente: En la primera línea se muestra el número de nodos (35). En cada una de las líneas restantes, se muestra información de cada uno de los nodos. La información se divide en campos que se describen a continuación: el primer valor es un identificador único del nodo o ID, los siguientes dos valores corresponden a los ID's de los nodos hijos tando del nodo que contesta de forma verdadera como el del nodo que contesta de forma falsa a la pregunta compuesta. El cuarto valor

corresponde al incremento en la probabilidad después de que el nodo en cuestión ha sido dividido usando la pregunta asociada. El quinto valor es el número de observaciones existentes en el nodo. Finalmente, el último campo corresponde a la estructura de la mejor pregunta compuesta donde el símbolo de admiración (!) representa el negado de la pregunta y el número delante del nombre representa el contexto con los valores explicados anteriormente. Las preguntas que se encuentran separadas por un espacio representan la conjunción de las preguntas y las que se encuentran separadas por paréntesis representan la disyunción de dichas preguntas.

Recuerde que el proceso que lleva a cabo `bldtree` se realiza solamente para un estado de cada fonema o trifonema así que se tiene que realizar para cada estado de cada fonema. Sin embargo, como los resultados son similares no se incluirán en este trabajo.

Podado de los árboles de decisión

Después de la ejecución de la aplicación `prunetree`, los nuevos árboles de decisión (ya recortados o podados) son almacenados con el mimo nombre y extensión de los árboles no recortados. El siguiente listado muestra parte del contenido del árbol de decisiones `A` del estado 0.

```
n_node 35
0 1 34 6.772724e+002 1.211703e+002 ((!QUESTION2 1))
1 2 5 5.827086e+002 1.051072e+002 ((QUESTION0_11_R 1 QUESTION12 -1))
2 3 4 3.073108e+002 2.280334e+001 ((!QUESTION14 -1))
3 - - 3.371019e+002 1.780334e+001
4 - - 2.055372e+002 5.000000e+000
5 6 7 5.259037e+002 8.230383e+001 ((QUESTION0_6_R 1 QUESTION2_3_L -1))
6 - - 3.659384e+002 1.106108e+001
7 8 33 3.585151e+002 7.124274e+001 ((!QUESTION0_19_R 1))
8 9 32 3.362329e+002 6.524274e+001 ((!QUESTION0_18_R 1))
9 10 13 3.558221e+002 5.918665e+001 ((!QUESTION2_8_L -1 !QUESTION0_17_R 1 !QUESTION18 -1 QUESTION15 -1)
  (!QUESTION2_8_L -1 !QUESTION0_10_R 1 QUESTION7 -1))
10 11 12 2.326758e+002 7.000000e+000 ((!QUESTION0_6_R 1))
11 - - 1.447965e+002 4.000000e+000
12 - - 1.884552e+002 3.000000e+000
13 14 27 2.916134e+002 5.218665e+001 ((QUESTION0 -1)(!QUESTION0_6_R 1 !QUESTION0_17_R 1 QUESTION2_8_L -1)
  (!QUESTION0 -1 !QUESTION0_6_R 1 !QUESTION0_17_R 1 !QUESTION2_8_L -1 !QUESTION15 -1))
14 15 26 2.784781e+002 4.018665e+001 ((!QUESTION0 -1))
15 16 25 2.061989e+002 3.418666e+001 ((!QUESTION15 -1))
16 17 24 1.875946e+002 2.418674e+001 ((!QUESTION0_16_R 1))
17 18 23 1.212821e+002 1.500006e+001 ((!WDBNDRY_E 0))
18 19 22 1.136834e+002 1.200006e+001 ((!WDBNDRY_B 0))
19 20 21 1.164219e+002 7.000002e+000 ((!QUESTION2 -1))
20 - - 3.554215e+001 3.000000e+000
21 - - 3.711638e+001 4.000002e+000
22 - - 6.452830e+001 5.000059e+000
23 - - 8.877192e+001 3.000000e+000
24 - - 1.250486e+002 9.186684e+000
25 - - 6.564415e+001 9.999912e+000
26 - - 1.207367e+002 6.000000e+000
27 28 31 2.070165e+002 1.200000e+001 ((!QUESTION0_17_R 1))
28 29 30 1.878156e+002 8.000000e+000 ((!WDBNDRY_E 0))
29 - - 1.661284e+002 4.000000e+000
30 - - 1.812863e+002 4.000000e+000
31 - - 1.351516e+002 4.000000e+000
32 - - 2.585435e+002 6.056084e+000
33 - - 2.051467e+002 6.000000e+000
34 - - 2.780843e+002 1.606318e+001
```

El formato como se muestran las preguntas compuestas es el mismo descrito previamente.

6.1.9. Creación del archivo de definición del modelo ligado CD

Lo que cabe destacar de la corrida de la aplicación `tiestate`, que se encarga de la creación del archivo de definición del modelo ligado CD, es lo siguiente:

```
A A I b 0 ((!QUESTION15 -1 !QUESTION5 -1)) -> n
((!WDBNDRY_E 0)) -> y
((!QUESTION0_15_R 1)) -> y
((!QUESTION0_0_R 1)) -> n
-> 68

A A I b 1 ((QUESTION5 -1)(!QUESTION5 -1 QUESTION0_14_R 1)
  (!QUESTION0_14_R 1 !QUESTION0_4_R 1 !QUESTION11 1 QUESTION17 -1)) -> n
((!QUESTION0_4_R 1 QUESTION11 1 QUESTION0_0_R 1)) -> n
((!QUESTION0_4_R 1)) -> y
((QUESTION0 -1)(!QUESTION0 -1 !WDBNDRY_B 0 QUESTION8 1 WDBNDRY_E 0)) -> n
((!WDBNDRY_B 0 QUESTION2_4_L -1 WDBNDRY_E 0)(!WDBNDRY_B 0 !QUESTION2_4_L -1)) -> n
((!WDBNDRY_B 0)) -> n
-> 85

A A I b 2 ((QUESTION10 1 QUESTION0_0_R 1)) -> n
((!QUESTION0_14_R 1 QUESTION5 -1)(!QUESTION0_14_R 1 !QUESTION5 -1 !QUESTION17 -1)) -> y
((!QUESTION5 -1)) -> y
((!QUESTION0_4_R 1)) -> y
((QUESTION10 1 WDBNDRY_E 0)) -> n
((!QUESTION2_4_L -1 !QUESTION0_6_R 1 WDBNDRY_E 0 !QUESTION0_2_R 1)
  (!QUESTION2_4_L -1 !QUESTION0_6_R 1 !WDBNDRY_E 0)) -> y
((!WDBNDRY_E 0)) -> y
((!WDBNDRY_B 0)) -> n
-> 116

A A U b 0 ((!QUESTION15 -1 !QUESTION5 -1)) -> n
((!WDBNDRY_E 0)) -> y
((!QUESTION0_15_R 1)) -> y
((!QUESTION0_0_R 1)) -> n
-> 68

A A U b 1 ((QUESTION5 -1)(!QUESTION5 -1 QUESTION0_14_R 1)
  (!QUESTION0_14_R 1 !QUESTION0_4_R 1 !QUESTION11 1 QUESTION17 -1)) -> n
((!QUESTION0_4_R 1 QUESTION11 1 QUESTION0_0_R 1)) -> n
((!QUESTION0_4_R 1)) -> y
((QUESTION0 -1)(!QUESTION0 -1 !WDBNDRY_B 0 QUESTION8 1 WDBNDRY_E 0)) -> n
((!WDBNDRY_B 0 QUESTION2_4_L -1 WDBNDRY_E 0)(!WDBNDRY_B 0 !QUESTION2_4_L -1)) -> n
((!WDBNDRY_B 0)) -> n
-> 85

A A U b 2 ((QUESTION10 1 QUESTION0_0_R 1)) -> n
((!QUESTION0_14_R 1 QUESTION5 -1)(!QUESTION0_14_R 1 !QUESTION5 -1 !QUESTION17 -1)) -> y
((!QUESTION5 -1)) -> y
((!QUESTION0_4_R 1)) -> y
((QUESTION10 1 WDBNDRY_E 0)) -> n
((!QUESTION2_4_L -1 !QUESTION0_6_R 1 WDBNDRY_E 0 !QUESTION0_2_R 1)
  (!QUESTION2_4_L -1 !QUESTION0_6_R 1 !WDBNDRY_E 0)) -> n
((!WDBNDRY_I 0 !WDBNDRY_B 0 QUESTION0_2_R 1 !QUESTION19 -1)) -> n
((!WDBNDRY_B 0)) -> n
-> 108
...

U SIL N b 0 ((!QUESTION0_0_R 1)) -> y
((!QUESTION2_7_L -1)) -> y
((QUESTION0_13_R 1)) -> y
```

```
((!QUESTION16 -1)) -> y
((!WDBNDRY_B 0)) -> n
-> 403

U SIL N b 1 ((!QUESTION2_7_L -1)) -> y
((QUESTION0_0_R 1)) -> n
((QUESTION0_13_R 1)) -> y
((!QUESTION16 -1)) -> y
((!WDBNDRY_B 0)) -> n
-> 408

U SIL N b 2 ((!QUESTION2_7_L -1)) -> y
((!QUESTION0_0_R 1)) -> y
((!QUESTION2_19_L -1)) -> y
((!QUESTION14 1)) -> y
((!WDBNDRY_B 0)) -> n
-> 415
```

Aquí se puede observar que estado de cada trifonema cuenta con un árbol de decisión recortado y se asigna a un senon después de haber ligado los estados. Su formato es el siguiente:

```
Trifonema p e rama1_árbol_podado -> respuesta
rama2_árbol_podado -> respuesta
...
ramaN_árbol_podado -> respuesta
-> senón
```

donde `Trifonema` son cada uno de los trifonemas que se obtuvieron del archivo del modelo de definición CD, `p` es el marcador de posición del trifonema, ya sea de acuerdo a la posición que ocupe en la palabra o si se trate de un solo trifonema (b, e, i, s); `e` corresponde a cada uno de los estados del trifonema que, en este caso toma los valores 0,1 o 2. Al estado le sigue cada una de las ramas del árbol podado, cuyos estados fueron ligados. Las ramas representan preguntas lingüísticas (`QUESTIONm_n_P`), negadas o no negadas, ó límites de palabra (`WDBNDRY_B`, `WDBNDRY_E`,etc.) . La respuesta a cada rama se puede dar en forma afirmativa (`y`) o negativa (`n`). Finalmente, `senón` corresponde al ID del senón que se ha creado.

En los resultados de la ejecución de la corrida puede presentarse también lo siguiente:

```
X E_1 I i 0  -> 417

X E_1 I i 1  -> 418

X E_1 I i 2  -> 419
```

que solamente indica que un estado de un trifonema dentro de una palabra es asignado a un determinado senón.

La salida del ejecutable `tiestate` se muestra a continuación:

```
n_node 35
0 8 1 6.664020e+002 1.550101e+002 ((!QUESTION15 -1 !QUESTION5 -1))
8 28 9 4.796702e+002 8.976432e+001 ((!QUESTION0_4_R 1 QUESTION11 1)(QUESTION0_14_R 1))
28 30 29 2.674786e+002 2.779237e+001 ((!QUESTION0_0_R 1))
30 32 31 1.533512e+002 1.200000e+001 ((QUESTION0_14_R 1))
32 34 33 1.358023e+002 7.000000e+000 ((!WDBNDRY_E 0))
34 - - 1.508738e+002 3.000000e+000
33 - - 2.031464e+002 4.000000e+000
```

```
31 - - 1.542161e+002 5.000000e+000
29 - - 2.463740e+002 1.579237e+001
9 11 10 4.226478e+002 6.197195e+001 ((!QUESTION0_4_R 1))
11 13 12 3.503697e+002 5.597195e+001 ((!QUESTION0_16_R 1))
13 17 14 3.053944e+002 4.519361e+001 ((QUESTION0_3_R 1)(!QUESTION0_3_R 1 QUESTION0_0_R 1 !QUESTION0 -1))
17 27 18 2.511930e+002 3.819361e+001 ((QUESTION15 1 !QUESTION19 -1))
27 - - 2.914178e+002 5.000000e+000
18 22 19 2.444714e+002 3.319361e+001 ((WDBNDRY_E 0 !QUESTION0_2_R 1 !QUESTION8 -1)(!WDBNDRY_E 0))
22 24 23 2.009593e+002 2.119355e+001 ((!WDBNDRY_B 0))
24 26 25 1.488990e+002 1.719355e+001 ((!WDBNDRY_E 0))
26 - - 1.560736e+002 7.000000e+000
25 - - 2.131317e+002 1.019355e+001
23 - - 1.273005e+002 4.000000e+000
19 21 20 2.182977e+002 1.200006e+001 ((!QUESTION0_2_R 1))
21 - - 2.611812e+002 8.999983e+000
20 - - 1.119188e+002 3.000075e+000
14 16 15 2.048183e+002 7.000000e+000 ((!WDBNDRY_B 0))
16 - - 1.406173e+002 3.000000e+000
15 - - 1.094495e+002 4.000000e+000
12 - - 2.948176e+002 1.077834e+001
10 - - 2.695907e+002 6.000000e+000
1 3 2 4.826471e+002 6.524575e+001 ((!WDBNDRY_E 0))
3 5 4 3.404296e+002 5.324830e+001 ((!QUESTION0_15_R 1))
5 7 6 2.519994e+002 4.124483e+001 ((!QUESTION0_0_R 1))
7 - - 1.062489e+003 3.824483e+001
6 - - 8.006883e+001 3.000000e+000
4 - - 4.298203e+002 1.200347e+001
2 - - 3.741414e+002 1.199745e+001
```

Observe que su formato es similar al ya explicado, sólo cambia que ahora se representan los estados ligados.

6.1.10. Inicialización y entrenamiento de los modelos ligados CD con Gaussianas mixtas

Inicialización plana de los modelos ligados CD para N Gaussianas

Como ya se mencionó en la sección 5.1.10 del capítulo 5, la inicialización plana de los modelos ligados CD para N Gaussianas se realiza mediante la aplicación `init_mixw` y como su corrida es similar a la ya explicada en la sección 6.1.6 no se mostrará en este documento. Cabe señalar que el entrenamiento se realizó con 1,2,4 y 8 Gaussianas.

Entrenamiento de los modelos ligados CD con Gaussianas mixtas

En forma similar al entrenamiento CI y no ligado CD también se utilizan las aplicaciones `bw` y `norm` para obtener los parámetros del entrenamiento ligado CD para cada Gaussiana. Razón por la cuál sólamente se mostrarán las probabilidades por trama y los criterios de convergencia para las iteraciones de las correspondientes Gaussianas.

Iteración	Probabilidad total por trama	Radio de convergencia
1	−1.77581131241715	–
2	10.273236853734	6.78509483631487
3	10.6674613345117	0.0383739308642945

Tabla 6.6: Criterios de convergencia para el entrenamiento de modelos ligados CD con 1 Gaussiana mixta.

Iteración	Probabilidad total por trama	Radio de convergencia
1	10.3424591250552	–
2	13.8207998232435	0.336316600929251
3	17.1289482987185	0.239360132393456
4	19.5830136986301	0.143270057046949
5	20.0640322580645	0.0245630507559755

Tabla 6.7: Criterios de convergencia para el entrenamiento de modelos ligados CD con 2 Gaussianas mixtas.

Iteración	Probabilidad total por trama	Radio de convergencia
1	19.7254220061865	–
2	24.461511268228	0.240100782662907
3	30.3341581970835	0.240077028130443
4	37.246133451171	0.227861119770651
5	38.2003756076005	0.0256198984434324

Tabla 6.8: Criterios de convergencia para el entrenamiento de modelos ligados CD con 4 Gaussianas mixtas.

Iteración	Probabilidad total por trama	Radio de convergencia
1	37.8235086168802	–
2	43.5028722934158	0.150154332165816
3	51.4455368979231	0.182577935335765
4	63.0895050817499	0.226335827866476
5	64.3176314626602	0.0194664133015293

Tabla 6.9: Criterios de convergencia para el entrenamiento de modelos ligados CD con 8 Gaussianas mixtas.

Incremento del número de Gaussianas en los modelos ligados CD

De la ejecución de la aplicación `inc_comp` , lo que cabe señalar es el siguiente segmento:

```
0:0(1.49e+002)
1:0(1.02e+002)
2:0(1.10e+002)
3:0(1.10e+001)
4:0(1.70e+001)
5:0(9.00e+000)
6:0(3.30e+001)
7:0(3.50e+001)
8:0(2.70e+001)
9:0(1.12e+002)
10:0(7.40e+001)
...
417:0(9.00e+000)
418:0(1.00e+001)
419:0(1.30e+001)
```

Las líneas anteriores tienen el siguiente formato:

$$id\_estado\_ligado\_más\_probable : id\_densidad\_más\_probable(valor\_máximo\_peso\_mixto)$$

donde `id_estado_ligado_más_probable` es el ID del estado al que se le asigna la mayor probabilidad de ocurrencia, `id_densidad_más_probable` es el ID del estado que tiene mayor probabilidad de ocurrencia y `valor_máximo_peso_mixto` es el valor del peso mixto máximo. Por ejemplo, `0:0(1.49e+002).`

Sin embargo, este formato sólo es para una densidad pero puede extenderse para N densidades, según corresponda, de la siguiente manera:

$$\begin{aligned}
&id_estado_ligado_más_probable : id_densidad_más_probable(valor_máximo_peso_mixto)\\
&id_estado_ligado_1_menos_probable : id_densidad_1_menos_probable(valor_1_peso_mixto)\\
&id_estado_ligado_2_menos_probable : id_densidad_2_menos_probable(valor_2_peso_mixto)\\
&\ldots\\
&id_estado_ligado_{N-1}_menos_probable : id_densidad_{N-1}_menos_probable(valor_{N-1}_peso_mixto)
\end{aligned}$$

donde `id_estado_ligado_más_probable` es el ID del estado al que se le asigna la mayor probabilidad de ocurrencia, `id_densidad_más_probable` es el ID del estado que tiene mayor probabilidad de ocurrencia, `valor_máximo_peso_mixto` es el valor del peso mixto máximo, $\texttt{id\_estado\_ligado}_\texttt{i}\texttt{\_menos\_probable} : \texttt{id\_densidad}_\texttt{j}\texttt{\_menos\_probable}$ es cada uno de los estados que tienen menos probabilidad de ocurrencia ordenados en forma decreciente. Por ejemplo, si $N = 4$ se podría tener:

`0:2(5.10e+001)0(3.51e+001)1(3.49e+001)3(3.11e+001)`

6.1.11. Resultados de la decodificación

El archivo de transcripción se muestra a continuación:

```
<S> <SIL> PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC01F)
<S> <SIL> PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC02F)
<S> <SIL> PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC03F)
<S> <SIL> PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC04F)
<S> <SIL> PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC05F)
<S> <SIL> PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC06F)
<S> <SIL> PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY <SIL> </S> (BC01F)
<S> <SIL> PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY <SIL> </S> (BC02F)
<S> <SIL> PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY <SIL> </S> (BC03F)
```

```
<S> <SIL> PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY <SIL> </S> (BC04F)
<S> <SIL> EN UN MOMENTO MAS LA INFORMACION VIAL <SIL> </S> (BC01F)
<S> <SIL> EN UN MOMENTO MAS LA INFORMACION VIAL <SIL> </S> (BC02F)
<S> <SIL> EN UN MOMENTO MAS LA INFORMACION VIAL <SIL> </S> (BC03F)
<S> <SIL> PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC01F)
<S> <SIL> PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC02F)
<S> <SIL> PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS <SIL> </S> (BC03F)
```

En el archivo de hipótesis de salida por palabras reconocidas se obtuvo lo siguiente:

```
PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC01F)
PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC02F)
PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC03F)
PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC04F)
PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC05F)
PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC06F)
PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY (WAV/BEATRIZC/DECODE/AUDITORIO/BC01F)
PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY (WAV/BEATRIZC/DECODE/AUDITORIO/BC02F)
PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY (WAV/BEATRIZC/DECODE/AUDITORIO/BC03F)
PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY (WAV/BEATRIZC/DECODE/AUDITORIO/BC04F)
EN UN MOMENTO MAS LA INFORMACION VIAL (WAV/BEATRIZC/DECODE/MOMENTO/BC01F)
EN UN MOMENTO MAS LA INFORMACION VIAL (WAV/BEATRIZC/DECODE/MOMENTO/BC02F)
EN UN MOMENTO MAS LA INFORMACION VIAL (WAV/BEATRIZC/DECODE/MOMENTO/BC03F)
PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F)
PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/INFORMATIVO/BC02F)
PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/INFORMATIVO/BC03F)
```

Parte de la corrida para la aplicación `sphinx3_livepretend` para la oración `WAV/BEATRIZC/DECODE/PARA/BC01F` arroja lo siguiente:

```
Backtrace(WAV/BEATRIZC/DECODE/PARA/BC01F)
FV:WAV/BEATRIZC/DECODE/PARA/BC01F>  WORD  SFrm  EFrm AScr(UnNorm) LMScore  AScr+LScr    AScale
fv:WAV/BEATRIZC/DECODE/PARA/BC01F> <SIL>     0     8     -97129   -74100    -171229     -31894
fv:WAV/BEATRIZC/DECODE/PARA/BC01F>  PARA     9    30   -1158854   -81586   -1240440    -483290
fv:WAV/BEATRIZC/DECODE/PARA/BC01F> OCHENTA  31    72   -2254812   -29279   -2284091    -830642
fv:WAV/BEATRIZC/DECODE/PARA/BC01F>     Y    73    75    -454950    -6052    -461002     -88057
fv:WAV/BEATRIZC/DECODE/PARA/BC01F>  OCHO    76    92    -683688    -4522    -688210    -383974
fv:WAV/BEATRIZC/DECODE/PARA/BC01F> PUNTO    93   129   -2225707    -4522   -2230229    -634767
fv:WAV/BEATRIZC/DECODE/PARA/BC01F> NUEVE   130   154   -2026442    -4522   -2030964    -449660
fv:WAV/BEATRIZC/DECODE/PARA/BC01F>NOTICIAS 155   204   -1973148    -4522   -1977670    -459030
fv:WAV/BEATRIZC/DECODE/PARA/BC01F> <SIL>   205   225    -393738   -74100    -467838    -110997
FV:WAV/BEATRIZC/DECODE/PARA/BC01F>              TOTAL -11268468  -283205

FWDVIT: PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATRIZC/DECODE/PARA/BC01F
)
FWDXCT: WAV/BEATRIZC/DECODE/PARA/BC01F S -3640632 T -7793948 A -7796157 L 2209 0
 -97129 -8101 <SIL> 9 -1158854 -8933 PARA 31 -2254812 -3121 OCHENTA 73 -454950 -
540 Y 76 -683688 -370 OCHO 93 -2225707 -370 PUNTO 130 -2026442 -370 NUEVE 155 -1
973148 -370 NOTICIAS 205 -393738 -8101 <SIL> 226 0 -22938 </S> 226
```

Se observa, en primer lugar, que los primeros resultados se presentan a manera de tabla y que tienen la palabra `fv:`, que indica el nombre y la ruta del archivo procesado. El significado de los demás encabezados se muestran en la tabla 6.10.

Después se observan el enunciado que comienza con `FWDVIT:` que indica el resultado por palabras reconocidas del algoritmo Viterbi, seguido del nombre del archivo procesado. Y en la la línea `FWDXCT:` se indican los resultados por puntajes de las palabras reconocidas en esta oración. Así para la oración `WAV/BEATRIZC/DECODE/PARA/BC01F` se obtuvo: el valor acústico escalado de la probabilidad acústica es -3640632, el resultado total para esta hipótesis es

Encabezado	Descripción
WORD	Indica la palabra que se reconoció.
SFrm	Indican la trama inicial donde se encontró la palabra.
EFrm	Indican la trama final donde se encontró la palabra.
AScr(UnNorm)	Probabilidad del modelo acústico sin escalar ($P(\mathbf{X}\|\mathbf{W})$). .
LMScore	Probabilidad del modelo del lenguaje ($P(\mathbf{W})$).
AScr + LScr	Indica la probabilidad de la secuencia de palabras dado el conjunto de observaciones ($P(\mathbf{W}\|\mathbf{X})$).
AScale	Factor de escala que se aplica al modelo acústico, tal y como se indicó en la sección 5.2 del capítulo 5.

Tabla 6.10: Encabezados de salida de `sphinx3_livepretend`.

−7793948, el resultado acústico total para esta hipótesis es −7796157, el resultado del modelo del lenguaje total para esta hipótesis es 2209.
Además, en el caso de la primera palabra reconocida se tiene: la trama de inicio para la palabra es la trama 0, el resultado acústico para la palabra es −97129, el resultado del modelo del lenguaje para la palabra es −8101 y corresponde a la palabra <SIL>.
Para las palabras restantes de la hipótesis se puede realizar un análisis similar para desglosar sus puntajes, lo cuál no se hará en este momento. Sólo se mencionará que se procesaron 226 tramas que es el campo final de la hipótesis de salida. Resultados similares se muestran a continuación para 3 ejemplos de las otras 3 frases restantes.

```
Backtrace(WAV/BEATRIZC/DECODE/AUDITORIO/BC01F)
FV:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> WORD     SFrm  EFrm AScr(UnNorm)  LMScore  AScr+LScr    AScale
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> <SIL>       0    46     1758725    -74100    1684625    1857883
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> PARA        7    64      115751    -81586      34165     279367
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> EL         65    71      233922    -39929     193993     257514
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F>AUDITORIO   72   119     1645719     -8246    1637473    1786613
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> QUE       120   129      271915     -8246     263669     315232
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> NOS       130   145      947771     -8246     939525    1005793
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F>ESCUCHA    146   185      502709     -8246     494463     677169
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F>  EN       186   191      197438     -8246     189192     233529
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F>  LA       192   204      669019     -8246     660773     722941
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> CIUDAD    205   253     2075993     -8246    2067747    2259990
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F>  DE       254   268      453260     -8246     445014     500784
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> MEXICO    269   306      433079     -8246     424833     625327
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> HAY       307   312       34595     -8246      26349      65682
fv:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F> <SIL>     313   343     1407759    -74100    1333659    1461162
FV:WAV/BEATRIZC/DECODE/AUDITORIO/BC01F>                 TOTAL  10747655    -352175

FWDVIT: PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY (WAV/BEATRI
ZC/DECODE/AUDITORIO/BC01F)
FWDXCT: WAV/BEATRIZC/DECODE/AUDITORIO/BC01F S 12644396 T -1296769 A -1301331 L 4
562 0 1758725 -8101 <SIL> 47 115751 -8933 PARA 65 233922 -4304 EL 72 1645719 -78
4 AUDITORIO 120 271915 -784 QUE 130 947771 -784 NOS 146 502709 -784 ESCUCHA 186
197438 -784 EN 192 669019 -784 LA 205 2075993 -784 CIUDAD 254 453260 -784 DE 269
 433079 -784 MEXICO 307 34595 -784 HAY 313 1407759 -8101 <SIL> 344 0 -20664 </S> 344

Backtrace(WAV/BEATRIZC/DECODE/MOMENTO/BC01F)
FV:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>  WORD     SFrm  EFrm AScr(UnNorm) LMScore  AScr+LScr  AScale
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F> <SIL>       0     8    416655     -74100     342555   457654
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F> <SIL>       9    30   1238402     -74100    1164302  1295849
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>   EN       31    37    110897     -97309      13588   159340
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>   UN       38    51    168720     -30191     138529   264850
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F> MOMENTO     52   100   1079017     -10279    1068738  1493875
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>  MAS       101   125    599568     -10279     589289   929501
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>   LA       126   143    673699     -10279     663420   736713
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>INFORMACION 144   230   1277186     -10279    1266907  1762422
```

```
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>  VIAL      231   246   649764     -10279     639485   818887
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>  <SIL>     247   256   306193     -74100     232093   332733
fv:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>  <SIL>     257   262   320336     -74100     246236   364841
FV:WAV/BEATRIZC/DECODE/MOMENTO/BC01F>                TOTAL  6840437    -475295

FWDVIT: EN UN MOMENTO MAS LA INFORMACION VIAL (WAV/BEATRIZC/DECODE/MOMENTO/BC01F
)
FWDXCT: WAV/BEATRIZC/DECODE/MOMENTO/BC01F S 9185513 T -1773223 A -1776228 L 3005
 0 416655 -8101 <SIL> 9 1238402 -8101 <SIL> 31 110897 -10680 EN 38 168720 -3222
UN 52 1079017 -1010 MOMENTO 101 599568 -1010 MAS 126 673699 -1010 LA 144 1277186
 -1010 INFORMACION 231 649764 -1010 VIAL 247 306193 -8101 <SIL> 257 320336 -8101
 <SIL> 263 0 -19879 </S> 263

Backtrace(WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F)
FV:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>  WORD     SFrm  EFrm AScr(UnNorm) LMScore AScr+LScr  AScale
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>  <SIL>       0    15  152887      -74100    78787    220576
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>  PARA        16    32  202109      -81586   120523    273498
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>PANORAMA     33    83 1346034      -47006  1299028   1546848
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>INFORMATIVO  84   157  479337      -10279   469058    715823
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F> OCHENTA    158   200   74781      -10279    64502    272281
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>    Y       201   203   84701      -10279    74422     98805
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>   OCHO     204   222  580671       -4522   576149    635088
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>   PUNTO    223   258  627146       -4522   622624    731425
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>   NUEVE    259   284  990136       -4522   985614   1055822
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F> NOTICIAS   285   345 2434523       -4522  2430001   2620065
fv:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F> <SIL>      346   364  246681      -74100   172581    313869
FV:WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F>                TOTAL 7219006    -325717

FWDVIT: PARA PANORAMA INFORMATIVO OCHENTA Y OCHO PUNTO NUEVE NOTICIAS (WAV/BEATR
IZC/DECODE/INFORMATIVO/BC01F)
FWDXCT: WAV/BEATRIZC/DECODE/INFORMATIVO/BC01F S 8885160 T -1262089 A -1265094 L
3005 0 152887 -8101 <SIL> 16 202109 -8933 PARA 33 1346034 -5090 PANORAMA 84 4793
37 -1010 INFORMATIVO 158 74781 -1010 OCHENTA 201 84701 -1010 Y 204 580671 -370 O
CHO 223 627146 -370 PUNTO 259 990136 -370 NUEVE 285 2434523 -370 NOTICIAS 346 24
6681 -8101 <SIL> 365 0 -22938 </S> 365
```

Así los resultados obtenidos para todas las frases se muestran en las tablas 6.11 y 6.12.

Descripción	Cantidad de frases	Porcentaje
elocuciones	15	100 %
Reconocidas correctamente	15	100 %
Con error	0	0 %
Con sustitución	0	0 %
Con inserción	0	0 %
Con eliminación	0	0 %

Tabla 6.11: Resultados del reconocedor de *Sphinx 3* para 25 frases en español.

Observe que SER = 0 %.
Observe que WER = 0 % por lo que WAC = 100 %.

6.1.12. Resultados del modelado del lenguaje

La ocurrencia de palabras en el archivo de transcripción utilizado para esta parte es:

```
AUDITORIO 4
CIUDAD 4
DE 4
```

```
EL 4
EN 7
ESCUCHA 4
HAY 4
INFORMACION 3
INFORMATIVO 3
LA 7
MAS 3
MEXICO 4
MOMENTO 3
NOS 4
NOTICIAS 9
NUEVE 9
OCHENTA 9
OCHO 9
PANORAMA 3
PARA 13
PUNTO 9
QUE 4
UN 3
VIAL 3
Y 9
```

Del modelo del lenguaje se obtuvieron: 26 unigramas, 31 bigramas y 25 trigramas. La perplejidad del lenguaje es de 1.75 con entropia de 0.81 bits, como se ve en la evaluación del modelo del lenguaje:

```
perplexity -text etc/Tesis_test.transcription
Computing perplexity of the language model with respect
   to the text etc/Tesis_test.transcription
Perplexity = 1.75, Entropy = 0.81 bits
Computation based on 138 words.
Number of 3-grams hit = 97  (70.29%)
Number of 2-grams hit = 25  (18.12%)
Number of 1-grams hit = 16  (11.59%)
80 OOVs (36.70%) and 0 context cues were removed from the calculation.
```

6.2. Reconocimiento de 109 palabras continuas del español hablado en la región central de México

Las corridas de las aplicaciones son similares a los de la sección 6.1 por lo que solamente me enfocaré a presentar los resultados pertinentes:

Descripción	**Cantidad de palabras**	**Porcentaje**
Palabras pronunciadas	138	100 %
Reconocidas correctamente	138	100 %
Con error	0	0 %
Con sustitución	0	0 %
Con inserción	0	0 %
Con eliminación	0	0 %

Tabla 6.12: Resultados del reconocedor de *Sphinx 3* a nivel de palabra para 25 frases en español.

6.2.1. Creación del archivo de definición del modelo

Se obtuvieron 29 fonemas base (`n_base`), ordenados alfabéticamente y considerando el silencio (`SIL`) como parte de ellos. 0 trifonemas (`n_tri`). 116 estados en total, 87 emitidos y 29 no emitidos, de los cuáles los emitidos están numerados del 0 al 86 y los no emitidos tienen el valor `N`. 87 estados ligados (`n_tied_state`), en este caso no se generaron estados ligados, así que son los mismos que el total de estados emitidos. 29 fonemas base después de haberlos compartido (`n_tied_ci_state`), en este caso no se compartieron, así que son los mismos que el total de estados emitidos. 29 matrices de transición (`n_tied_tmat`), numeradas del 0 al 28.

6.2.2. Inicialización plana de los parámetros CI

Inicialización de pesos mixtos y matrices de transición

Las matrices de transición se escribieron en un arreglo de 29 matrices con 3 estados emitidos 1 no emitido ($[3 \times 4]$) cada una. Los pesos mixtos se escribieron en un arreglo de dimensiones $[87 \times 1 \times 1]$, que corresponden a la cantidad de estados emitidos, flujos de características empleados y cantidad de densidades, respectivamente.

Inicialización de la media y la variancia globales

Acumulación y normalización para la media global

Se escribió la media global acumulada en el archivo `gauden_counts` como un arreglo de dimensiones $[1 \times 1 \times 1]$. Se normalizaron los valores leídos para una mezcla Gaussiana, un vector de características y una densidad y se escribieron los valores normalizados al archivo de media global `globalmean`.

Acumulación y normalización para la variancia global

Se escribieron los valores de la variancia global acumulada en el archivo `gauden_counts` como un arreglo de dimensiones $[1 \times 1 \times 1]$. Se indica que se normalizaron los valores leídos para una mezcla Gaussiana, un vector de características y una densidad y se escribieron los valores normalizados al archivo de variancia global `globalvar`.

Copia de la media y variancia globales a cada estado HMM

Para la copia de la media global se leyó el arreglo de dimensiones $[1 \times 1 \times 1]$ del archivo de la media global y se guardaron las copias en el archivo de medias para los estados `means`como un arreglo de dimensiones $[87 \times 1 \times 1]$ (87 estados emitidos, 1 flujo de características y 1 densidad).
Para la copia de la variancia global se leyó el arreglo de dimensiones $[1 \times 1 \times 1]$ del archivo de la variancia global y se guardaron las copias en el archivo de

variancias para los estados `variances`como un arreglo de dimensiones $[87 \times 1 \times 1]$ (87 estados, 1 flujo de características y 1 densidad).

6.2.3. Entrenamiento de los modelos independientes del contexto (CI)

En este caso la convergencia se dió en 11 iteraciones. Las probabilidades totales por trama y los radios de convergencia se muestran en la tabla 6.13.

Iteración	Probabilidad total por trama	Radio de convergencia
1	−7.58454989062763	–
2	−6.14544337876493	0.189741847916516
3	−3.03039374053508	0.506887696499433
4	−1.16510466094565	0.615526970848373
5	−0.657184418643783	0.435943876397865
6	−0.475172976611139	0.276956417208212
7	−0.352079757698132	0.259049283044017
8	−0.307661113915531	0.12616074287544
9	−0.278933030455999	0.0933757376547097
10	−0.265683156654888	0.0475019892031075
11	−0.256907790678109	0.0330294403577046

Tabla 6.13: Criterios de convergencia para el entrenamiento de modelos CI.

6.2.4. Creación del archivo de definición del modelo no ligado dependiente del contexto

Para el caso del archivo de la definición del modelo no ligado CD que contiene a todos los trifonemas se obtuvieron en este archivo, además de los 29 fonemas base, 2636 trifonemas, 10660 estados en total, 7995 emitidos y 2665 no emitidos, de los cuáles los emitidos están numerados del 0 al 7994 y los no emitidos tienen el valor `N`. 7995 estados ligados (`n_tied_state`). 87 fonemas base después de haberlos compartido (`n_tied_ci_state`). 29 matrices de transición (`n_tied_tmat`), numeradas del 0 al 28.

Para el caso del archivo de la definición del modelo no ligado CD con la lista corta de trifonemas se obtuvieron 29 fonemas base, 587 trifonemas, 2464 estados en total, 1848 emitidos y 616 no emitidos, de los cuáles los emitidos están numerados del 0 al 1847 y los no emitidos tienen el valor `N`. 1848 estados ligados (`n_tied_state`). 87 fonemas base después de haberlos compartido (`n_tied_ci_state`). 29 matrices de transición (`n_tied_tmat`), numeradas del 0 al 28.

6.2.5. Inicialización plana de los parámetros no ligados dependientes del contexto

Se calcularon los parámetros iniciales para el modelo CD, alojando en memoria el espacio necesario para cada arreglo de los parámetros no ligados CD y se escribieron los respectivos archivos de matrices de transición (`transition_matrices` [$29 \times 3 \times 4$ `array`]), los pesos mixtos (`mixture_weights` [$1848 \times 1 \times 1$ `array`]), las medias (`means` [$1848 \times 1 \times 1$ `array`]) y las variancias (`variances` [$1848 \times 1 \times 1$`array`]).

6.2.6. Entrenamiento de los modelos no ligados CD

En este caso la convergencia se dió en 4 iteraciones. Las probabilidades totales por trama y los radios de convergencia se muestran en la tabla 6.14.

Iteración	Probabilidad total por trama	Radio de convergencia
1	−0.248722025912839	–
2	15.3680901901397	62.7882157148607
3	34.4861517751977	1.24401024125459
4	35.3315328958439	0.0245136403202327

Tabla 6.14: Criterios de convergencia para el entrenamiento de modelos no ligados CD.

6.2.7. Construcción de los árboles de decisión para compartir parámetros

Generación de las preguntas lingüísticas

Las preguntas lingüísticas que se generaron para cada estado del modelo no ligado CD fueron: 20 preguntas lingüísticas a patir del estado 0, 21 preguntas lingüísticas a patir del estado 1 y 21 preguntas lingüísticas a patir del estado 2.

Podado de los árboles de decisión

La cantidad de senones existente antes de realizar el recorte fue de 1554, el número de nodos terminales fue de 230 y el número de nodos recortados fue de 554.

6.2.8. Creación del archivo de definición del modelo ligado CD

Del proceso de ligado de estados se crearon 1087 senones.

6.2.9. Inicialización y entrenamiento de los modelos ligados CD con Gaussianas mixtas

Inicialización plana de los modelos ligados CD para N Gaussianas

Como ya se mencionó en la sección 5.1.10 del capítulo 5, la inicialización plana de los modelos ligados CD para N Gaussianas se realiza mediante la aplicación `init_mixw` y como su corrida es similar a la ya explicada en la sección 6.1.6 no se mostrará en este documento. Cabe señalar que el entrenamiento se realizó con 1,2,4 y 8 Gaussianas.

Entrenamiento de los modelos ligados CD con Gaussianas mixtas

En las siguientes tablas sólamente se mostrarán las probabilidades por trama y los criterios de convergencia para las iteraciones de las correspondientes Gaussianas.

Iteración	Probabilidad total por trama	Radio de convergencia
1	−0.248722025912839	–
2	20.832643446071	84.7587397803341
3	22.3358404846037	0.0721558472607679
4	22.8370351674239	0.0224390339448223

Tabla 6.15: Criterios de convergencia para el entrenamiento de modelos ligados CD con 1 Gaussiana mixta.

Iteración	Probabilidad total por trama	Radio de convergencia
1	22.4920915362611	–
2	25.5537775534242	0.136122779521288
3	28.3909641595154	0.111028070122298
4	31.2564866229177	0.100930790772103
5	31.5164226821471	0.00831622767988194

Tabla 6.16: Criterios de convergencia para el entrenamiento de modelos ligados CD con 2 Gaussianas mixtas.

Iteración	Probabilidad total por trama	Radio de convergencia
1	31.1801615345785	–
2	34.431280498065	0.104268830034155
3	38.1097257277469	0.106834401058324
4	41.8117953895339	0.0971423853384389
5	42.2764092209322	0.0111120277679967

Tabla 6.17: Criterios de convergencia para el entrenamiento de modelos ligados CD con 4 Gaussianas mixtas.

Iteración	Probabilidad total por trama	Radio de convergencia
1	41.9382466767626	–
2	46.1757529867071	0.103631714773285
3	52.0147736833249	0.101041570540725
4	59.4900891805485	0.143715236419843
5	60.3293454484267	0.0141074972224555

Tabla 6.18: Criterios de convergencia para el entrenamiento de modelos ligados CD con 8 Gaussianas mixtas.

Incremento del número de Gaussianas en los modelos ligados CD

Esta sección se omite debido a que es similar a la sección 6.1.10.

6.2.10. Resultados de la decodificación

Los resultados que se muestran a continuación fueron obtenidos con una densidad Gaussiana. Cabe aclarar que se decodificó también con 8 densidades Gaussianas, sin embargo, como los resultados de decodificación fueron muy pobres, fueron omitidos. El archivo de transcripción es:

```
<S> <SIL> TODA LA REPUBLICA QUE TENGAN UN EXCELENTE JUEVES ++HEM++ Y QUE DESAYUNEN MUY RICO <SIL> </S>
 (AP02F)
<S> <SIL> OCHO DE LA MAÑANA CON UN MINUTO <SIL> TOMA DE ESCUELA POR PADRES DE FAMILIA EN DONDE <SIL> </S>
 (AP03F)
<S> <SIL> OCHO DE LA MAÑANA CON DOS MINUTOS CONTINUA LA INVESTIGACION DE UN HOMBRE QUE(2) ASESINO POR LA
 MAÑANA <SIL> </S> (AP05F)
<S> <SIL> TU ESTAS EN LA ESCUELA EN DONDE(2) PUES ++HMM++ SUCEDIERON ESTOS HECHOS LAMENTABLES QUE NOS
 TIENES BUENOS DIAS <SIL> </S> (AP06F)
<S> ++HEM++ GRACIAS SON OCHO DE LA MAÑANA CON CUATRO MINUTOS EXIGE LA COMISION PERMANENTE DEL CONGRESO
 UNA EXPLICACION POR EL CASO <SIL> </S> (AP07F)
<S> <SIL> OCHO DE LA MAÑANA CON CUATRO MINUTOS <SIL> </S> (AP08F)
<S> <SIL> OCHO DE LA MAÑANA CON CINCO MINUTOS GARANTIZA EL PRESIDENTE <SIL> </S> (AP09F)
<S> <SIL> PROTECCION A LOS JOVENES <SIL> ESTUDIANTES <SIL> </S> (AP10F)
<S> <SIL> DURANTE LA CUMBRE EL PRESIDENTE <SIL> </S> (AP11F)
<S> EXHORTO A DELEGADOS Y PRESIDENTES MUNICIPALES DE DIVERSOS PARTIDOS A PLANEAR EL CRECIMIENTO URBANO
 DE LO CONTRARIO ++HEM++ EL PAIS ENFRENTARA PROBLEMAS ++HEM++ DE SERVICIOS Y SEGURIDAD SON LAS OCHO DE
 LA MAÑANA CON SEIS MINUTOS CRECE EL DESCONTENTO MINERO CON LA SECRETARIA DEL TRABAJO <SIL> </S> (AP12F)
<S> <SIL> ESTADOUNIDENSE EXPLICA <SIL> COMO EVITAR UNA CRISIS EN PETROLEOS MEXICANOS <SIL> </S> (AP13F)
<S> ++HEM++ OCHO DE LA MAÑANA CON SIETE MINUTOS SI USTED PLANEA VIAJAR A EUROPA <SIL> </S> (AP14F)
<S> ++HEM++ YA NO SERA REQUERIDO EL PASAPORTE DE CIUDADANOS NORTEAMERICANOS PARA INGRESAR A MEXICO <SIL>
 </S> (AP15F)
```

Se ocuparon distintos modelos del lenguaje con los mismos elementos entrenados. Se trabajo con dos técnicas de descuento en el modelo del lenguaje: Good Turing y Witten Bell.

Al usar el modelo del lenguaje con *Good Turing* en el archivo de hipótesis de salida por palabras reconocidas se obtuvo lo siguiente:

```
OCHO DE LA DE CRECE DE PARA LA Y CRECE DE DEL (WAV/ARELIP/DECODE/AP02F)
OCHO DE LA MAÑANA CON UN (WAV/ARELIP/DECODE/AP03F)
OCHO DE MAÑANA CON DOS (WAV/ARELIP/DECODE/AP05F)
OCHO DE LA ESCUELA (WAV/ARELIP/DECODE/AP06F)
LA DOS EN OCHO DE LA MAÑANA CON CUATRO MINUTOS Y LA CON (WAV/ARELIP/DECODE/AP07F)
OCHO DE LA MAÑANA CON CUATRO MINUTOS (WAV/ARELIP/DECODE/AP08F)
OCHO DE LA MAÑANA CON (WAV/ARELIP/DECODE/AP09F)
OCHO PRESIDENTES PRESIDENTES (WAV/ARELIP/DECODE/AP10F)
```

```
OCHO DE LA CON PRESIDENTE (WAV/ARELIP/DECODE/AP11F)
ESTOS PARA LA LA DOS PRESIDENTES DE (WAV/ARELIP/DECODE/AP12F)
OCHO (WAV/ARELIP/DECODE/AP13F)
OCHO DE LA MAÑANA CON (WAV/ARELIP/DECODE/AP14F)
ESTAS (WAV/ARELIP/DECODE/AP15F)
```

Así los resultados obtenidos para todas las palabras se muestran en las tablas 6.19 y 6.20.

Descripción	**Cantidad de frases**	**Porcentaje**
elocuciones	13	100 %
Reconocidas correctamente	1	7.69 %
Con error	12	92.31 %
Con sustitución	8	61.54 %
Con inserción	4	30.77 %
Con eliminación	12	92.31 %

Tabla 6.19: Resultados del reconocedor de *Sphinx 3* para 13 oraciones en español usando Good Turing.

Para SER se tiene $SER = \frac{1}{13} * 100\,\% \ = 7.69\,\%$

Descripción	**Cantidad de palabras**	**Porcentaje**
Palabras pronunciadas	189	100 %
Reconocidas correctamente	43	22.75 %
Con error	146	77.25 %
Con sustitución	29	15.34 %
Con inserción	4	2.12 %
Con eliminación	117	61.94 %

Tabla 6.20: Resultados del reconocedor de *Sphinx 3* a nivel de palabra para 109 palabras en español usando Good Turing.

Para WER se tiene

$$WER_{Good\ Turing} = \frac{29 + 4 + 117}{189} = 79.36\,\% \tag{6.3}$$

por lo que

$$WAC_{Good\ Turing} = (1 - 0.7936) * 100\,\% = 20.64\,\% \tag{6.4}$$

Con la estrategia de descuento *Witten Bell* se obtuvo lo siguiente:

```
OCHO DE LA DE CRECE EL PARA LA Y CRECE LA DEL (WAV/ARELIP/DECODE/AP02F)
OCHO DE LA MAÑANA CON UN EN LO PLANEA (WAV/ARELIP/DECODE/AP03F)
OCHO DE MAÑANA CON DOS MINUTOS CON CRECE (WAV/ARELIP/DECODE/AP05F)
OCHO DE LA ESCUELA EN EN DE PRESIDENTE ESTOS DE (WAV/ARELIP/DECODE/AP06F)
LA ESTAS EN OCHO DE LA MAÑANA CON CUATRO MINUTOS Y LA CON (WAV/ARELIP/DECODE/AP07F)
OCHO DE LA MAÑANA CON CUATRO MINUTOS (WAV/ARELIP/DECODE/AP08F)
OCHO DE LA MAÑANA CON (WAV/ARELIP/DECODE/AP09F)
OCHO PRESIDENTES PRESIDENTES (WAV/ARELIP/DECODE/AP10F)
OCHO DE LA CON PRESIDENTE (WAV/ARELIP/DECODE/AP11F)
ESTOS PARA LA LA DOS PRESIDENTES DE (WAV/ARELIP/DECODE/AP12F)
OCHO (WAV/ARELIP/DECODE/AP13F)
OCHO DE LA MAÑANA CON (WAV/ARELIP/DECODE/AP14F)
ESTAS (WAV/ARELIP/DECODE/AP15F)
```

Así los resultados obtenidos para todas las palabras se muestran en las tablas 6.21 y 6.22.

Descripción	Cantidad de frases	Porcentaje
elocuciones	13	100 %
Reconocidas correctamente	1	7.69 %
Con error	12	92.31 %
Con sustitución	10	76.92 %
Con inserción	6	46.15 %
Con eliminación	11	84.62 %

Tabla 6.21: Resultados del reconocedor de *Sphinx 3* para 13 oraciones en español usando Witten-Bell.

Para SER se tiene $SER = \frac{1}{13} * 100\,\% \ = 7.69\,\%$

Descripción	Cantidad de palabras	Porcentaje
Palabras pronunciadas	189	100 %
Reconocidas correctamente	47	24.87 %
Con error	142	75.13 %
Con sustitución	34	17.99 %
Con inserción	6	3.17 %
Con eliminación	115	60.85 %

Tabla 6.22: Resultados del reconocedor de *Sphinx 3* a nivel de palabra para 109 palabras en español usando Witten-Bell.

Para WER se tiene

$$WER_{Witten\ Bell} = \frac{34 + 6 + 115}{189} = 82.01\,\% \tag{6.5}$$

por lo que

$$WAC_{Witten\ Bell} = (1 - 0.8201) * 100\,\% = 17.99\,\% \tag{6.6}$$

6.2.11. Resultados del modelado del lenguaje

La ocurrencia de palabras en el archivo de transcripción utilizado para esta parte es:

```
A 5
ASESINO 1
BUENOS 1
CASO 1
CINCO 1
CIUDADANOS 1
COMISION 1
COMO 1
CON 8
CONGRESO 1
CONTINUA 1
CONTRARIO 1
```

```
CRECE 1
CRECIMIENTO 1
CRISIS 1
CUATRO 2
CUMBRE 1
DE 14
DEL 2
DELEGADOS 1
DESAYUNEN 1
DESCONTENTO 1
DIAS 1
DIVERSOS 1
DONDE 1
DONDE(2) 1
DOS 1
DURANTE 1
EL 7
EN 4
ENFRENTARA 1
ESCUELA 2
ESTADOUNIDENSE 1
ESTAS 1
ESTOS 1
ESTUDIANTES 1
EUROPA 1
EVITAR 1
EXCELENTE 1
EXHORTO 1
EXIGE 1
EXPLICA 1
EXPLICACION 1
FAMILIA 1
GARANTIZA 1
GRACIAS 1
HECHOS 1
HOMBRE 1
INGRESAR 1
INVESTIGACION 1
JOVENES 1
JUEVES 1
LA 14
LAMENTABLES 1
LAS 1
LO 1
LOS 1
MAÑANA 8
MEXICANOS 1
MEXICO 1
MINERO 1
MINUTO 1
MINUTOS 6
MUNICIPALES 1
MUY 1
NO 1
NORTEAMERICANOS 1
NOS 1
OCHO 7
PADRES 1
PAIS 1
PARA 1
PARTIDOS 1
PASAPORTE 1
PERMANENTE 1
PETROLEOS 1
PLANEA 1
PLANEAR 1
POR 3
PRESIDENTE 2
```

```
PRESIDENTES 1
PROBLEMAS 1
PROTECCION 1
PUES 1
QUE 3
QUE(2) 1
REPUBLICA 1
REQUERIDO 1
RICO 1
SECRETARIA 1
SEGURIDAD 1
SEIS 1
SERA 1
SERVICIOS 1
SI 1
SIETE 1
SON 2
SUCEDIERON 1
TENGAN 1
TIENES 1
TODA 1
TOMA 1
TRABAJO 1
TU 1
UN 3
UNA 2
URBANO 1
USTED 1
VIAJAR 1
Y 3
YA 1
```

Del modelo del lenguaje con Good Turing se obtuvieron: 112 unigramas, 159 bigramas y 167 trigramas. La perplejidad del lenguaje es de 58243433460581198000000000000000000000000000000000000000000000000000000000000000000000000.00 con entropia de 245.04 bits, como se ve en la evaluación del modelo del lenguaje:

```
evallm :
perplexity -text etc/Tesispal_test.transcription
Computing perplexity of the language model with respect
   to the text etc/Tesispal_test.transcription
Perplexity = 58243433460581198000000000000000000000000000000000000000000000000000000
0000000.00, Entropy = 245.04 bits
Computation based on 189 words.
Number of 3-grams hit = 150  (79.37%)
Number of 2-grams hit = 19  (10.05%)
Number of 1-grams hit = 20  (10.58%)
71 OOVs (27.31%) and 0 context cues were removed from the calculation.
```

Del modelo del lenguaje con Witten Bell se obtuvieron: 112 unigramas, 159 bigramas y 167 trigramas.La perplejidad del lenguaje es de 1.75 con entropia de 0.81 bits, como se ve en la evaluación del modelo del lenguaje:

```
 perplexity -text etc/Tesispal_test.transcription
Computing perplexity of the language model with respect
   to the text etc/Tesispal_test.transcription
Perplexity = 3.20, Entropy = 1.68 bits
Computation based on 189 words.
Number of 3-grams hit = 150  (79.37%)
Number of 2-grams hit = 19  (10.05%)
Number of 1-grams hit = 20  (10.58%)
71 OOVs (27.31%) and 0 context cues were removed from the calculation.
```

7

Conclusiones, mejoras y aplicaciones

Conclusiones

1. En cuanto al reconocimiento de frases para el idioma español se logró un 100 % de reconocimiento exitoso ($Word\ Error\ Rate = 0\,\%$). Lo que prueba la versatilidad del sistema *Sphinx* . El hecho de que haya resultado esta cantidad se debe al tamaño del vocabulario (25 palabras) y a lo restringida que resultó la gramática. También a que en este caso el etiquetado que se hizo permitió mejorar la precisión del reconocedor.

2. En cuanto a las 109 palabras empleadas en reconocimiento de voz continua para el idioma español se emplearon dos estrategias de descuento en el modelado del lenguaje: Good Turing y Witten Bell. La estrategia Good Turing proporcionó una mejor precisión de palabra ($WAC_{Good\ Turing} = 20.64\,\%$) que la estrategia Witten Bell ($WAC_{Witten\ Bell} = 7.99\,\%$), sin embargo, se obtuvo un mayor porcentaje de palabras reconocidas exitosamente con la estrategia Witten Bell (24.87 %) que con la estrategia Good Turing (22.75 %). Esto se debe en parte a la perplejidad del lenguaje ya que para Witten Bell es mucho menor que para Good Turing. Por falta de tiempo no pude realizar el modelado del lenguaje con otras estrategias de descuento (Lineal y Absoluta) e incluso utilicé los valores por omisión de las estrategias Good Turing y Witten Bell, lo cuál requeriría más tiempo para obtener los valores de descuento adecuados a este grupo de palabras para mejorar la precisión del reconocedor de *Sphinx*.

3. A medida que el número de palabras incrementa la complejidad de la gramática aumenta y por lo mismo la posibilidad de que se realice un reconocimiento preciso disminuye. Asimismo, la poca cantidad de datos ocasiona que se vea mermada la potencia del reconocedor, tal fue el caso que se me presentó cuando utilicé 8 densidades Gaussianas, pues al

particionar las densidades, la dispersión de los datos fue demasiada. Se espera que para un vocabulario de 10000 palabras o más, las densidades Gaussianas realmente ayuden al proceso de decodificación.

4. Durante el proceso de decodificación de las 109 palabras empleadas en reconocimiento de voz continua para el idioma español el mismo *Sphinx Decoder* eliminó un conjunto muy importante de senones (620) que, de acuerdo con este software, "no se inicializaron" lo que desde mi punto de vista contribuyó a obtener una pobre precisión de palabra.

5. El reconocimiento de voz continua presenta un grado de complejidad muy alto, ya que para emplear *Sphinx* se debe poseer de un conocimiento profundo de este tema puesto que, en ocasiones, los autores de la documentación, dan muchas cosas por sobreentendidas y sólo se dedican a describir los aspectos más esenciales. Además se deben dominar la programación tanto en Perl como en Lenguaje C, que son los lenguajes empleados en el código fuente de *Sphinx 3*, *SphinxTrain* y *CMU-SLM*.

6. Las limitantes más importantes a las que me enfrenté en la realización de esta investigación consisten en que la documentación de *Sphinx* no se encuentra bien organizada, ni actualizada; en algunos casos se encuentra incompleta o repetida e incluso, a veces, los mismos autores desconocen su aplicación. Además, se debe estar atento para no perderse en un mar de documentos en línea que a veces complican más el entendimiento del sistema, puesto que cualquier información acerca de *Sphinx* se encuentra distribuida en varias páginas web de usuarios experimentados y desarrolladores. Lo mismo sucede con el código fuente.

   Como primer ejemplo tenemos en el tutorial de *Sphinx* , Robust group's Open Source Tutorial, tiene una sección que indica cómo llevar a cabo un entrenamiento preliminar, sin embargo, en este documento se hace alusión a un módulo que ya no existe y que sigue estando como parte de la descripción del proceso de entrenamiento preliminar. Dicho segmento de código es:

   ```
   perl scripts_pl/40.buildtrees/slave.treebuilder.pl
   perl scripts_pl/45.prunetree/slave-state-tying.pl
   ```

   cuyo módulo `45.prunetree/slave-state-tying.pl` ya no existe, puesto que el módulo `40.buildtrees/slave.treebuilder.pl` es quién realiza ambas tareas.

   También como ejemplo se tiene al documento The Hieroglyphs, que pretende reunir toda la documentación de *Sphinx 3* , se encuentra incompleto, ya que el mismo autor reconoce que no sabe el funcionamiento de ciertas aplicaciones como `init_mix` y por lo mismo se encuentra sin

terminar.

Otro ejemplo es que existen dos páginas distintas del manual del usuario de *Sphinx 3*: una en http://www.speech.cs.cmu.edu/sphinxman/ y otra en http://cmusphinx.sourceforge.net/sphinx3/doc/. Difieren solamente en los dos primeros artículos en donde el primero hace alusión a repaso de la teoría de estimación por máxima similitud y distinción de HMM discretos, continuos y semicontinuos, mientras que en la segunda se hace referencia a las distintas versiones del decodificador de *Sphinx* y la descripción del decodificador *s*3.6. En los demás artículos son los mismos documentos bajo nombres distintos, lo que que ocasiona confusión.

El siguiente ejemplo consiste en que a veces para obtener información acerca de un tema en determinados documentos, se hace referencia a los documentos de las versiones anteriores del decodificador de *Sphinx* , por lo que a veces resulta confuso saber sobre cuál versión están hablando, ya que existen diferencias significativas entre todas ellas. En particular cuando traté revisar algunos detalles en la manipulación de los árboles de léxico de la versión *s*3.6 del decodificador fuí remitido a una presentación de la versión *s*3.2 que me aclaró la duda, pero que considero que toda la información pertinente debería estar almacenada en el mismo documento.

Otro ejemplo que causa confusión se puede observar en el documento Troubleshooting: tools and logfiles, con la explicación que se da acerca de la ejecución correcta de la aplicación `bldtree`. En ella no sólo se describe el significado de los resultados de la corrida de la aplicación sino que los mezclan con la explicación del algoritmo en sí, lo que desde mi punto de vista debería de estar en la documentación de Instruction set for training.

7. Aunque ya se ha trabajado con *Sphinx* en México aún no se ha desarrollado un sistema de reconocimiento de voz continua del español por gente de nuestro país, ya que el entrenamieto y la decodificación solamente se han realizado mediante las aplicaciones que incluye *Sphinx*. A pesar de no haber terminado el sistema de reconocimiento elaboré varias herramientas para el manejo de *Sphinx*. En la tabla 7.1 se muestran las herramientas que creé y también se describe su utilidad.

Mejoras

1. Estructurar un nivel de etiquetado más fino, ya que solamente se empleó un nivel de etiquetado básico debido a que el proceso de etiquetado es muy complejo y necesita de gran cantidad de tiempo para llevarse a cabo.

2. Formar módulos más especializados que no requieran de volver a realizar los mismos cálculos una y otra vez.

Herramienta	Utilidad
`creadic`	Herramienta que, a partir de un archivo de transcripción, permite crear en forma automática los diccionario de palabras y de unidades acústicas para el idioma español.
`createOperationsMap`	Herramienta que se encarga de crear el archivo de mapa de operaciones que define, por un lado, a los estados destino a los que se desea copiar la media o la variancia globales y, por otro, a los estados de origen que contienen la media y variancia que se desea copiar.
`getRatio`	Herramienta que se encarga de obtener el radio de convergencia entre las probabilidad total de la iteración actual y la probabilidad total de la iteración previa que se realizan con los programas `bw`(algoritmo de Baum-Welch) y `norm` (normalización). Esta herramienta permite establecer una razón de convergencia para que se detengan las iteraciones en cuanto se cumpla dicha razón o se alcance un número máximo de iteraciones.
`creaarboles`	Herramienta que me permite generar el conjunto de árboles de decisión para el ligado de estados del modelo CD. `bldtree` solamente permite generar un árbol de decisión por cada estado de cada unidad acústica por lo que fue necesario crear esta herramienta con el fin de construir todos los árboles de decisión para cada estado de cada unidad acústica.
`InicializaNGaus`	Herramienta que se encarga de copiar los modelos ligados CD anteriores, que se obtuvieron en el proceso de dividir las densidades Gaussianas, a las nuevas modelos ligados CD. Se detiene en cuanto se alcance el número de Gaussianas que se quieren entrenar.

Tabla 7.1: Herramientas creadas para el entrenamiento de *Sphinx 3* .

3. Realizar en un solo módulo el algoritmo Baum-Welch y la normalización ya que estos procesos van uno a continuación del otro en cada entrenamiento, por lo que es posible unirlos sin afectar el desempeño de *Sphinx*.

4. En el caso de las 109 palabras del español, cambiar el número de senones de entrenamiento definido para mejorar la precisión del reconocimiento, ya que con vocabularios pequeños se requiere que el número de éstos sea mucho menor al preestablecido (1000 senones).

Posibles aplicaciones

- En automatización de sistemas que requieran del reconocimiento de voz del español, tales como software de dictado, sistemas de reservaciones de una agencia de viajes, software para telefonía celular, control de iluminación de una casa, etc.
- En sistemas multimodales, para recibir instrucciones e interactuar con el usuario, como un edificio inteligente o un robot.

- En minería de datos para la extracción de información relevante acerca de una conversación o un discurso.

En fin, considero que el número de aplicaciones es vasto, el límite, por supuesto, se encuentra en nuestra imaginación.

Apéndice A

Etiquetado de la parte del habla

A.1. Parte del habla

Los lingüistas agrupan las palabras en clases o conjuntos que muestran un comportamiento sintáctico similar y, frecuentemente, un tipo semántico típico. Tales clases de equivalencias se denominan tradicionalmente como *parte del habla* (*part of speech, POS*), *clases de palabras*, *clases morfológicas* o *etiquetas léxicas*.

La parte del habla para una palabra proporciona una cantidad de información significativa acerca de las palabras y las palabras circundantes. Tres elementos importantes en la parte del habla son *sustantivos*, *verbos* y *adjetivos*. Los sustantivos generalmente se refieren a las personas, animales, conceptos o cosas; los verbos se utilizan para expresar la acción en una oración y los adjetivos describen propiedades acerca de los sustantivos. Existen otros elementos, como los pronombres posesivos (mío, tuyo, suyo, etc) y los pronombres personales (yo, tú, él, etc.). El saber si una palabra es de un tipo determinado indica qué palabras son más probables de ocurrir en su vecindad. Lo cuál puede ser muy útil en el modelado del lenguaje para el reconocimiento de voz.

La parte del habla también indica algo acerca de cómo se pronuncia una determinada palabra y proporciona su significado, por ejemplo *término* es sustantivo y *terminó* es un verbo. De esta manera, la parte del habla permite generar palabras pronunciadas en forma más natural en sistemas de síntesis de voz y permite mayor precisión en sistemas de reconocimiento de voz.

Asimismo, la parte del habla también puede ser usada en recuperación de información para obtener la forma subyacente de una palabra cuando se le añaden inflecciones. En este caso, el conocer la parte del habla de una palabra puede

ayudar a saber qué afijos morfológicos puede tomar dicha palabra. Además puede ayudar en aplicaciones de recuperación de información para extraer palabras importantes en un documento, como los sustantivos o los verbos. Existen etiquetadores automáticos de la parte del habla que pueden ayudar en la construcción de algoritmos automáticos que eliminen la ambigüedad en las palabras de un corpus y también existen etiquetadores de la parte del habla que se utilizan en el modelo del lenguaje de sistemas automáticos de reconocimiento de voz avanzados y que se basan en *N-gramas*, como se discutirá en el capítulo 3. La parte del habla también se utiliza frecuentemente en el análisis sintáctico de textos en forma parcial, por ejemplo, para encontrar rápidamente los nombres o frases en aplicaciones de extracción de información.

A.2. Etiquetado

La meta más importante en el procesamiento del lenguaje natural (Natural Language Processing, NLP) es analizar y comprender el lenguaje. La investigación en NLP se enfoca en tareas intermedias que hacen que le dan sentido a la estructura inherente del lenguaje sin requerir de un total entendimiento. Una de las tareas es el *etiquetado de la parte del habla* (*POS tagging*) o simplemente *etiquetado* [26]. El etiquetado es la tarea poner un distintivo a cada palabra en una oración con su apropiada parte del habla, es decir, nosotros asociamos cada palabra con su correspondiente categoría; decimos si corresponde a un sustantivo, un verbo, un adjetivo, etcétera. En la tabla A.1 se muestra la identificación de las categorías léxicas o parte del habla de las palabras del español definido en el corpus DIME [28].

El etiquetado es un caso limitado de análisis sintáctico para eliminar la ambigüedad. Debido a que muchas palabras tienen más de una categoría sintáctica, el etiquetado trata de determinar cuál de estas categorías sintácticas es la más probable para el uso particular de una palabra dentro de una oración.

El etiquetado es un problema de alcance limitado: En lugar de construir un análisis sintáctico completo, sólo se mantienen fijas las categorías sintácticas de las palabras en una oración. Por lo que el etiquetado es mucho más fácil de resolver que el análisis sintáctico y su precisión es bastante alta (entre el 96 y 97 %). Sin embargo, es importante notar que esta precisión admirable no es del todo cierta, ya que se hace con base a un análsis por palabras.

A.2.1. Fuentes de información en el etiquetado

En el etiquetado surge una pregunta muy importante: ¿Cómo se puede decidir que está utilizando la parte del habla correcta en un contexto determinado? Existen dos fuentes de información que pretenden dar solución a esta respuesta:

- *información sintagmática*. Una forma es analizando las etiquetas de otras palabras en el contexto de la palabra que nos interesa. Estas palabras también pueden ser ambiguas en sus respectivas partes del habla, pero la observación esencial es que algunas secuencias de partes del habla son más

Etiqueta	Categoría
N	Sustantivo
V	Verbo
VAM	Verbo Auxiliar-Modal
VC	Verbo con Clítico
A	Adjetivo
AD	Adjetivo Demostrativo
TD	Artículo Determinado
TI	Artículo Indefinido
R	Adverbio
RI	Adverbio Interrogativo
RR	Adverbio Relativo
RN	Adverbio de Negación
RA	Adverbio de Afirmación
P	Pronombre
PD	Pronombre Demostrativo
PR	Pronombre Relativo
PI	Pronombre Interrogativo
PC	Pronombre Clítico
S	Preposición
C	Conjunción

Tabla A.1: Parte del habla del español

comúnes o más probables que otras, por ejemplo, la secuencia **TD N V** (`artículo` – `determinado sustantivo verbo`). Este tipo de información sintagmática estructural es la más obvia pero no es la más útil, debido a que muchas palabras en el español pueden tener muchas partes del habla.

- *información léxica*. Tomando en cuenta que la palabra involucrada da una gran cantidad de información acerca de la etiqueta correcta. La información léxica es bastante útil porque la distribución en el uso de una palabras en diferentes partes del habla generalmente es en extremo desigual.

Todos los etiquetadores automáticos modernos de alguna manera utilizan una combinación de información sintagmática, al buscar información acerca de las secuencias de etiquetas, y de información léxica, tratando de predecir una etiqueta basado en lo que concierne a las palabras.
Se utilizan aproximaciones estadísticas para indicar que una palabra tiene una determinada probabilidad de pertenecer a una categoría de la parte del habla en particular, pero también existe otra probabilidad de que pertenezca a alguna otra categoría. Esto sugiere la utilización de modelos ocultos de Markov para determinar tal probabilidad.

A.2.2. Etiquetado manual

Aunque se está tratando con la parte del habla, el etiquetado también se puede extender a nivel de alófonos y a nivel de fonemas, es decir se plantean distintos niveles de detalle para mejorar la precisión del reconocimiento. Para

facilitar el proceso de etiquetado manual se utilizó el la herramienta *Praat*, versión 5.0.09, de los autores Paul Boersma y David Weenink de la Universidad de Amsterdan [1]. *Praat* es un software que permite analizar, sintetizar y manipular señales de voz, así como crear ilustraciones de estas señales para tesis y artículos científicos. Asimismo *Praat* permite visualizar el oscilograma, el espectrograma y el pitch de la señal de voz que se está analizando, entre otras cosas. En la figura A.1 podemos observar la forma en cómo se realiza el proceso de etiquetado en forma manual para la parte del habla y los fonemas de la frase: "Para ochenta y ocho punto nueve noticias". Obsérvese que se analiza el espectrograma para delimitar cada nivel. El tamaño de la ventana utilizada para el espectrograma es de 25 ms y se añaden al inicio y al final de cada oración silencios de aproximadamente 20 ms de duración.

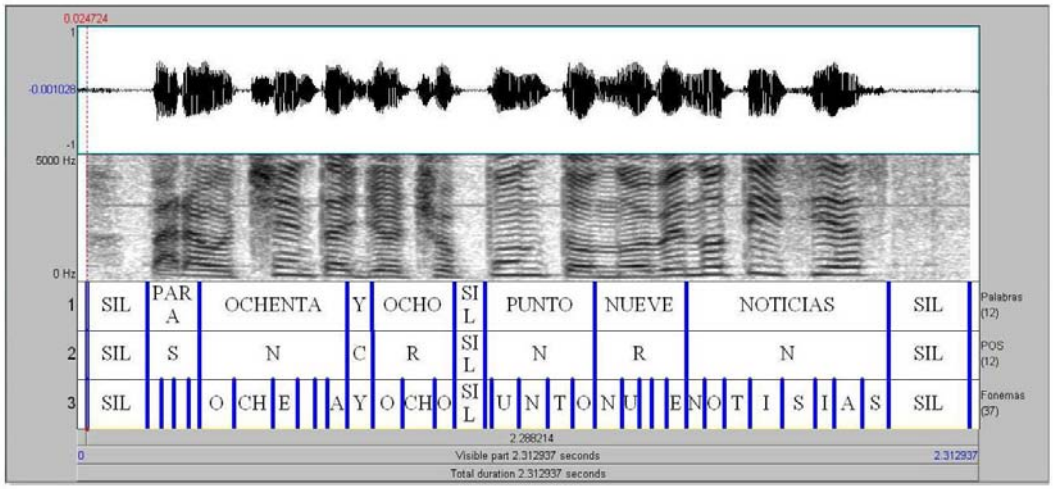


Figura A.1: Ejemplo de etiquetado de la parte del habla a nivel de palabras y de fonemas.

La herramienta *Praat* también permite generar archivos de etiquetado. Un segmento de un archivo de etiquetado para la frase anterior se muestra a continuación:

```
File type = "ooTextFile"
Object class = "TextGrid"

xmin = 0
xmax = 2.3129375
tiers? <exists>
size = 3
```

[1] http://www.fon.hum.uva.nl/praat/

```
item []:
    item [1]:
        class = "IntervalTier"
        name = "Palabras"
        xmin = 0
        xmax = 2.3129375
        intervals: size = 12
        intervals [1]:
            xmin = 0
            xmax = 0.0247236587854071
            text = ""
        intervals [2]:
            xmin = 0.0247236587854071
            xmax = 0.17891149407440596
            text = "SIL"
        intervals [3]:
            xmin = 0.17891149407440596
            xmax = 0.31360990795841814
            text = "PARA"
        intervals [4]:
            xmin = 0.31360990795841814
            xmax = 0.6924442739010805
            text = "OCHENTA"
        .
        .
        .
    item [2]:
        class = "IntervalTier"
        name = "POS"
        xmin = 0
        xmax = 2.3129375
        intervals: size = 12
        intervals [1]:
            xmin = 0
            xmax = 0.0247236587854071
            text = ""
        intervals [2]:
            xmin = 0.0247236587854071
            xmax = 0.17891149407440596
            text = "SIL"
        intervals [3]:
            xmin = 0.17891149407440596
            xmax = 0.31360990795841814
            text = "S"
        intervals [4]:
            xmin = 0.31360990795841814
```

```
            xmax = 0.6924442739010805
            text = "N"
        .
        .
        .
    item [3]:
        class = "IntervalTier"
        name = "Fonemas"
        xmin = 0
        xmax = 2.3129375
        intervals: size = 37
        intervals [1]:
            xmin = 0
            xmax = 0.0247236587854071
            text = ""
        intervals [2]:
            xmin = 0.0247236587854071
            xmax = 0.17891149407440596
            text = "SIL"
        intervals [3]:
            xmin = 0.17891149407440596
            xmax = 0.21600635651305197
            text = "P"
        intervals [4]:
            xmin = 0.21600635651305197
            xmax = 0.24681526130262438
            text = "A"
        intervals [5]:
        .
        .
        .
```

El parámetro `File type` presenta el tipo de archivo que se manipula, en este caso el archivo es de texto. El parámetro `Object class` hace referencia a la clase de objeto que se maneja en *Praat*, en este caso el objeto se denomina "TextGrid". Los valores de `xmin` y `xmax` de las líneas 4 y 5 indican el inicio y el fin de la señal de voz en segundos. El parámetro `tiers?` pregunta si existen niveles definidos que se usen para etiquetar, en este caso el valor $< exists >$ indica que sí existen tales niveles. De ser así, el parámetro `size` indica cuántos de estos niveles existen y se muestra en el siguiente renglón un arreglo denominado `item`, que es un arreglo global donde se definen los niveles y cuyo índice mínimo es 1. Observe que el arreglo `item` externo no tiene una dimensión establecida, lo que indica que se pueden definir tantos niveles de etiquetado como se requieran. A su vez este arreglo global contiene otros arreglos anidados también llamados `item`, donde se establecen, ahora,

los intervalos que se encuentran definidos en cada nivel. Para cada nivel se establece la clase a la que pertenecen, en este caso se trata de otra clase que maneja *Praat*, la clase es "IntervalTier". Además, cada arreglo `item` anidado lleva asociado un nombre que permita distinguir a un nivel de otro. En este ejemplo tenemos tres niveles: `Palabras`, `POS` y `Fonemas`. Asimismo cada arreglo `item` anidado cuenta con segmentos `xmin` y `xmax`, que indican también el inicio y el fin de cada intervalo que se está etiquetando, también en segundos. Ahora bien, cada intervalo cuenta una etiqueta denominada `intervals`, que define la sección de intervalos que tiene cada nivel. En cada sección el parámetro `size` indica el número de intervalos en que se divide cada nivel. Cada sección cuenta con un arreglo denominado `intervals,`cuyo índice también empieza con 1 y cuyos contenidos también cuentan con sus propios campos `xmin` y `xmax`, que indican también el inicio y el fin de cada segmento, más un campo denominado `text`, que tiene la etiqueta asociada con ese segmento. De esta manera se pueden establecer claramente los límites entre cada etiqueta de cada palabra de la POS y entre cada etiqueta de cada fonema de cada palabra con el fin de mejorar el modelado acústico y la precisión del decodificador.

Obsérvese también que en la figura A.1 para el etiquetado a nivel de fonemas se consideraron solamente los fonemas básicos del español[2]tratando de incluir también acentos. Sin embargo, debido a que la duración del corpus es de 5 horas y que en el momento en que se realizó el proceso de etiquetado se realizó en forma manual no fue posible realizar niveles de etiquetado más detallados, como se describen en [33].

[2]Para mayor referencia consulte [5].

Apéndice B

Documentación básica de *Sphinx 3*

Los documentos que encontré y que considero relevantes en cuanto a información técnica acerca de *Sphinx* se enlistan a continuación:

Speech at CMU Web Page Es la página de la Universidad de Carnegie Mellon dedicada a la investigación, desarrollo y 'despliegue' de tecnologías de voz. Es el punto de entrada a las tecnologías de voz que maneja esta Universidad, entre ellas *Sphinx* .

The CMU Sphinx Group Open Source Speech Recognition Engines Página principal del *Sphinx Group Project*. Contiene la documentación general de *Sphinx*, los componentes de *Sphinx*, las distintas versiones de los motores de reconocimiento de *Sphinx* que existen y algunas ligas externas a sitios de procesamiento de voz. Las secciones de documentación que destacan son el tutorial de *Sphinx*, el manual de *Sphinx* y la sección de wiki de la documentación.

Robust group's Open Source Tutorial Página que contiene un pequeño tutorial que muestra el proceso de configuración básica de los componentes de *Sphinx*. Además, contiene una breve introducción para aprender a utilizar *Sphinx* y llevar a cabo un entrenamiento y un reconocimiento prelimares. Como apéndices contiene referencias básicas de trifonemas, HMMs, ligado de estados y el modelo del lenguaje.

Manual for the Sphinx-III recognition system Reune 6 documentos escritos por Rita Singh. Contiene una breve explicación del método de estimación por máxima similitud, la diferencia entre HMM discretos, continuos y semicontinuos, el set de instrucciones para el entrenamiento, una sección de preguntas más frecuentemente realizadas (FAQ), una sección de solución de posibles problemas y una sección sobre algunos tópicos de

decodificación. Debido a su relevancia, los últimos 4 documentos también se incluirán en esta lista.

Instruction set for training (Sphinx-3 trainer/decoder manual) Contiene la serie de procedimiento necesarios que se deben de llevar a cabo tanto para el entrenamiento de modelos continuos como el de modelos semicontinuos con *SphinxTrain*. Describe en forma detallada las aplicaciones que se emplean en cada etapa así como sus parámetros. También incluye una sección de los requirimientos necesarios y consideraciones que se deben de tomar en cuenta antes de comenzar el entrenamiento.

Sphinx-3 FAQ Documento de preguntas más frecuentemente realizadas (FAQ). Contiene una serie de preguntas, junto con sus respuestas, que los usuarios de *Sphinx 3* podrían hacerse antes, durante y después del entrenamiento o de la decodificación.

Troubleshooting: tools and logfiles Documento acerca de la solución de posibles problemas. Muestra la ejecución correcta de cada herramienta que se utiliza tanto en el entrenamiento de modelos continuos como en el de modelos semicontinuos. Asimismo, presenta los posibles problemas, junto con sus soluciones, así como los errores y advertencias que pueden ocurrir durante la ejecución de dichas herramientas. Sin embargo, esta documentación está incompleta.

Sphinx-3 miscellaneous (Decoding) Contiene algunos tópicos acerca de la decodificación. Explica brevemente temas como la generación de celosías y de la lista N-mejor, así como de los resultados de las corridas de las aplicaciones asociadas con este procedimiento. También incluye la explicación del formato en el que se encuentra el archivo del modelo del lenguaje después de ser obtenido con SLM. Otros temas son la generación de hipótesis y la explicación de algunas banderas del decodificador de *Sphinx* 2.

CMU Sphinx documentation Wiki Contiene una wiki dividida en dos secciones principales: documentación para los usuarios y documentación para desarrolladores e investigadores. La sección de documentación de usuario contiene una subwiki de *Sphinx* , una sección cómo hacer un modelado del lenguaje sencillo con *Sphinx* , una sección de cómo adaptar los modelos acústicos por omisión a la voz del usuario y como adaptar el decodificador para que sea más rápido. En la sección de documentación para desarrolladores e investigadores describe aspectos particulares del código fuente de *Sphinx* y también aspectos que sirven de guía a los desarrolldores para llevar a cabo tareas de reconocimiento de voz que deseen.

Sphinx subwiki Documentación acerca de información del sistema de reconocimiento *Sphinx* haciendo énfasis en *Sphinx 3* . Describe con mayor precisión cada aspecto relevante de *Sphinx* . Tiene información concreta acerca de la utilización del decodificador *Sphinx 3* . Muestra solamente los

pasos generales que se deben seguir desde la grabación de las señales de audio, pasando por la descarga y configuración de *SphinxBase* hasta llegar a la instalación y configuración de los decodificadores `sphinx3_livedecode`, `sphinx3_livepretend` y `PocketSphinx`. También contiene tutoriales y documentación para desarrolladores, que cubre los aspectos de modelado acústico, modelado de lenguaje, elaboración de gramáticas de estados finitos, el diccionario de pronunciación que se usa en *CMU Sphinx* y bases de datos de audio, entre otros.

Hello World Decoder QuickStart Guide Una guía completa que muestra, paso a paso, cómo utilizar el decodificador de *Sphinx* . Una vez realizada la configuración del decodificador, se detallan los pasos necesarios para llevar a cabo la ejecución de los decodificadores; es equivalente al manual del usuario para la decodificación. Esta guía me fue de mucha utilidad ya que en otra documentación no existe tal descripción.

Sphinx 3 Internals Documentation Contiene las funciones, los archivos y las variables internas de los reconocedores de *Sphinx 3* . Su enfoque está más orientado a los programadores que a los usuarios de *Sphinx*.

SPHINX III Signal Processing Front End Specification Describe brevemente el front-end del procesamiento de señales de voz del sistema de reconocimiento de voz *Sphinx 3* . Se centra en las etapas sin profundizar más de lo necesario. Considero que este es otro de los pocos documentos bien organizados y concisos.

Sphinx Decoders Sirve para conocer las distintas versiones del decodificador decodificadores que ha realizado *Sphinx Group*.

Sphinx-3 $s3.X$ **Decoder** $(X = 6)$ Página que describe la versión actual del decodificador de *Sphinx 3* . Desglosa sus parámetros, sus recursos y describe la forma en que opera el decodificador.

The CMU-Cambridge Statistical Language Modeling Toolkit v2 (CMU-SLM) Página que contiene al conjunto de herramientas para el modelado del lenguaje. Contiene todos los aspectos de estas herramientas, desde su instalación, la terminología, los formatos de archivos, el desglose de la aplicación de cada herramienta con sus respectivos parámetros, la forma de uso y las estrategias de descuento que se pueden usar en los modelos del lenguaje. Considero que éste es uno de los pocos documentos que se encuentra bien organizado, es conciso y trata los puntos esenciales que se requieren a la hora de diseñar el modelo del lenguaje.

The Hieroglyphs Documento que trata de agrupar todo el conocimiento acerca de *Sphinx 3* , *SphinxTrain* y *CMU-SLM* y sus recursos relacionados para construir aplicaciones de voz. Es un documento bien explicado y detalla información pertinente acerca del uso de *Sphinx 3* , *SphinxTrain* y *CMU-SLM*, sin embargo, se encuentra incompleto.

Sphinx-II User Guide Manual del usuario de *Sphinx 2*. Entre otras cosas contiene la explicación del funcionamiento de la API de *Sphinx 2*. Es útil sólo como referencia, ya que en determinados documentos de *Sphinx 3* se hace referencia a *Sphinx 2*, pero no a éste manual. En esta documentación también se encuentra una sección acerca de cómo se debe elaborar una gramática de estados finitos para usarse en el reconocedor. Para trabajar con pequeños vocabularios, mi intención era la de desarrollar una gramática de éste tipo y así mejorar la precisión del reconocimiento, sin embargo, al no haber encontrado referencia alguna en el momento en que elaboré las gramáticas para este trabajo decidí utilizar gramáticas basadas en 3-gramas, construidas con las herramientas SLM.

Sourceforge Sphinx Página donde se pueden descargar todas las versiones de *Sphinx* y de los recursos asociados con él.

Sphinx Presentation Existen una serie de diapositivas que se incluyen con la documentación de *Sphinx 3*, las cuáles hacen referencia a las mejoras entre distintas versiones. Me ayudaron a elegir cuál versión del decodificador de *Sphinx* usar. Así, elegí la versión *s*3.6 porque tiene:

- Mejor arquitectura de búsqueda.
- Mayor soporte en la adaptación de locutores.
- Máyor portabilidad y mejor creación del código ejecutable cuando se compila en otra plataforma.
- Soporte y seguridad para gramáticas de estados finitos.

El código fuente de las aplicaciones de *SphinxTrain*, *Sphinx Decoder* y *CMU-SLM* En estos archivos se encuentran, además de los derechos de autor y las revisiones que se le han hecho a cada software, comentarios acerca de lo qué hace cada programa. Algunos contienen comentarios específicos acerca de los elementos esenciales de cómo se programó y otros no, por lo que fue necesario leer las líneas de código para entender dichos programas, sin embargo, la notación y uso de las variables son bastante descriptivas lo que facilitó su comprensión. Aún así, el código en ocasiones no sólo es extenso sino que la cantidad de archivos que tuve que revisar también requirieron de muchas horas de análisis y si se carece de los fundamentos necesarios tanto de Procesamiento de voz continua como de programación en lenguaje C resulta bastante difícil.

Sphinx Knowledge Base Tool Página de *CMU Sphinx Group* que contiene una herramienta para obtener el modelo del lenguaje para el sistema *Sphinx* directamente. El inconveniente es que dicha herramienta trabaja con el conjunto de fonemas base del idioma inglés lo cuál no fue útil para los propósitos de esta tesis, sin embargo, me permitió resolver algunas dudas acerca de qué estrategia de descuento usar.

Apéndice C

Flujos de características de *Sphinx*

Al conjunto de coeficientes dinámicos que se han generado por medio del front-end de *Sphinx*[1] se le denomina *conjunto de vectores de características base* o *flujo de características base*. Este conjunto puede ser extendido para incluir características de más alto orden, que representan los cambios acústicos de la señal. Algunas extensiones comunes que se realizan a los vectores son los coeficientes delta cepstral y doble delta cepstral[2]. *Sphinx* también maneja *vectores de características delta de término largo* (*long-term delta*) y difieren de los vectores delta (en tiempo corto) en que sólo capturan las tendencias sobre una gran ventana de tiempo.
En modelos continuos y semicontinuos es necesario especificar cuantos flujos de características se desean usar y la manera en que se desean ordenar. Existen dos conjuntos de notación para representar el mecanismo de generación de coeficientes dinámicos, una para el decodificador y otra para el entrenador, respectivamente: la notación genérica y la notación simplificada. De hecho se establece una correspondencia uno a uno entre ambas notaciones. La tabla C.1 muestra ambas notaciones. Los flujos que utiliza *Sphinx 3* son $4s\_12c\_24d\_3p\_12dd$, para modelos semicontinuos compatibles con *Sphinx 2*, $1s\_12c\_12d\_3p\_12dd$, para modelos continuos que manejan los coeficientes de potencia separados de los coeficientes cepstral, y $1s\_c\_d\_dd$, para modelos continuos que manejan los coeficientes de potencia junto con los coeficientes cepstral. Los demás flujos de características ya son obsoletos.

En ambas notaciones la c hace alusión a los coeficientes cepstral, la d hace referencia a los coeficientes delta cepstral y la dd indica los coeficiente doble delta cepstral. En la notación genérica el 0 indica el coeficiente de potencia, ya sea cepstral, delta cepstral o doble delta cepstral, de acuerdo al símbolo que se anteponga antes de la diagonal, /; el rango $0..L-1$ hace referencia al coeficiente

[1] Para conocer el front-end de Sphinx refiérase a [41].
[2] Véase [3].

Notación genérica	**Notación simplificada**
$c/1..L-1/, d/1..L-1/, c/0/d/0/dd/0/, dd/1..L-1/$	$4s\_12c\_24d\_3p\_12dd$
$c/1..L-1/d/1..L-1/c/0/d/0/dd/0/dd/1..L-1/$	$1s\_12c\_12d\_3p\_12dd$
$c/0..L-1/d/0..L-1/dd/0..L-1/$	$1s\_c\_d\_dd$
$c/0..L-1/d/0..L-1/$	$1s\_c\_d$
$c/0..L-1/$	$1s\_c$
$c/0..L-1/dd/0..L-1/$	$1s\_c\_dd$

Tabla C.1: Notaciones para los flujos de características de *Sphinx*.

de potencia junto con todos los coeficientes, ya sea cepstral, delta cepstral o doble delta cepstral de acuerdo al símbolo que se anteponga antes de la diagonal, /; el rango $1..L-1$ hace referencia solamente a los coeficientes, ya sea cepstral, delta cepstral o doble delta cepstral de acuerdo al símbolo que se anteponga antes de la diagonal, /; y la coma sirve como separador de características. En la notación simplificada, el número que se le antepone a la s indica el número de vectores de características que se trabajarán en paralelo y que no se especifican en su notación genérica pero que así también se les considera; y la p indica que se trata del coeficiente de potencia manejado en forma separada; en caso de no incluir la p significa que el coeficiente de potencia se maneja en forma conjunta con los coeficientes, ya sea cepstral, delta cepstral o doble delta cepstral, según se especifique con la letra que le sigue.
Así al analizar el flujo de características $4s\_12c\_24d\_3p\_12dd$ hacemos uso de su notación genérica:

$$c/1..L-1/, d/1..L-1/, c/0/d/0/dd/0/, dd/..L-1/$$

se tiene que se generaron 4 vectores con los que se trabajará en paralelo. Además se puede deducir que dimensionalidad de cada vector es de 51 al descomponerlo en sus partes de la siguiente manera:

$$\begin{array}{cccccccccc} c/1..L-1/, & & d/1..L-1/, & & c/0/d/0/dd/0/, & & dd/1..L-1/ & = \\ 12\ cepstral & + & 24\ delta & + & \begin{array}{c}3\ términos\\ de\ potencia\end{array} & + & 12\ doble\ delta & = \\ 12\ cepstral & + & \begin{array}{c}12\ deltas\\ +\\ 12\ deltas\ de\\ término\ largo\end{array} & + & \begin{array}{c}3\ términos\\ de\ potencia\end{array} & + & 12\ doble\ delta & = 51 \end{array}$$

En forma similar se puede deducir que para $1s\_c\_d\_dd$, que es el utilizado en este proyecto, se trabaja con un vector de características y su dimensionalidad es de 39 debido a que al aplicar los criterios mencionados en su notación genérica

se tiene:

$$\begin{array}{lll} c/0..L-1/ & d/0..L-1/ & dd/0..L-1/ \\ 13\ cepstral + & 13\ delta + & 13\ doble\ delta \\ \begin{array}{c} 1\ coeficiente \\ de\ potencia \\ + \\ 12\ cepstral \end{array} + & \begin{array}{c} 1\ coeficiente \\ delta\ de \\ potencia \\ + \\ 12\ deltas \end{array} + & \begin{array}{c} 1\ coeficiente \\ doble\ delta \\ de\ potencia \\ + \\ 12\ doble\ delta \end{array} \end{array} \begin{array}{l} = \\ = \\ \\ \\ = 39 \end{array}$$

Apéndice D

Archivos necesarios para el entrenamiento de *Sphinx 3*

Antes de realizar el proceso de entrenamiento en el sistema *Sphinx* se requieren de los siguientes elementos de apoyo [45]:

- Un conjunto de *archivos de características* (*feature files*) o *flujos de características* calculados previamente con los datos de audio para el entrenamiento, cada archivo representa una grabación que se tiene en el cuerpo del entrenamiento. Las grabaciones puede ser transformado en una secuencia de vectores de características usando el ejecutable `wav2feat` proporcionado en la arquitectura de software de *SphinxTrain* y cuyos parámetros se muestran en la tabla E.6 del apéndice E. Por ejemplo, en este trabajo para el cálculo de los flujos de características de los archivos que se encuentran en `Tesis_train.fileids` y que tienen formato `raw`, que se encuentran en el directorio `wav`, que tienen extensión `pcm`, que se van a guardar en el directorio `feat`, con la extensión `mfc`,con frecuencia de muestreo de 16 kHz, con frecuencias mínima y máxima del banco de filtros de 133.33334 Hz y 6855.4976, respectivamente y que utilizan 40 filtros triangulares, se utilizó el siguiente comando:

```
bin/wave2feat -verbose yes -c ./Tesis_train.fileids -raw yes \
 -di wav -ei pcm -do feat -eo mfc -srate 16000.0 -lowerf 133.33334 \
-upperf 6855.4976 -nfilt 40
```

  Un segmento de un archivo de características se muestra a continuación:

$$
\begin{array}{l}
226 frames \\
\begin{array}{rrrrrrrrrr}
11.851 & -0.916 & -0.660 & -0.203 & -0.215 & 0.454 & -0.148 & -0.183 & 0.179 & -0.081 \\
12.079 & -0.707 & -0.379 & -0.291 & -0.300 & 0.208 & -0.264 & -0.301 & 0.085 & -0.065 \\
11.728 & -0.594 & -0.405 & -0.211 & 0.004 & 0.317 & -0.138 & -0.034 & 0.204 & 0.046 \\
11.168 & -0.423 & -0.432 & -0.348 & -0.253 & 0.287 & -0.129 & -0.224 & 0.287 & 0.098
\end{array} \\
\vdots
\end{array}
$$

  donde *frames* indica en número de tramas que forman a la señal y los valores de cada renglón representan un segmento de los coeficientes MFCC calculados.

- Un *archivo de control* (*control file*) que contiene una lista de nombres de archivos y la ruta completa de los vectores de características. Un ejemplo del contenido de este archivo para el entrenamiento es el siguiente:

```
BeatrizC/TRAIN/PARA/BC01
BeatrizC/TRAIN/PARA/BC02
BeatrizC/TRAIN/PARA/BC03
...
```

  Estos archivos no llevan las extensiones. Éstas serán proporcionadas separadamente en el transcurso del entrenamiento, debido a que se desea dar nombres únicos a todos los archivos, de igual forma se incluye la dirección completa parecida para hacer cada entrada única en el archivo de control.

- Un *archivo de transcripción* (*transcript file*) en cual se listan la transcripción textual correspondiente a los archivos de características, exactamente en el mismo orden como se listan los nombres de los archivos de características en el archivo de control. Por ejemplo, un segmento del archivo de transcripción utilizado en este trabajo es:

```
< S > < SIL > PARA OCHENTA Y OCHO PUNTO NUEVE NOTICIAS < SIL > < /S > (BC01)
< S > < SIL > PARA EL AUDITORIO QUE NOS ESCUCHA EN LA CIUDAD DE MEXICO HAY < SIL > < /S > (BC02)
...
```

- *léxico* o *diccionario de pronunciación*: Un archivo de léxico de pronunciación o diccionario especifica las pronunciaciones de las palabras que son de interés para el entrenador y el decodificador. Por ejemplo, para una parte del vocabulario utilizado en este trabajo el diccionario es:

```
AUDITORIO   A U D I T O R I O
CIUDAD      S I U D A D
DE          D E
EL          E L
EN          E N
ESCUCHA     E S K U CH A
HAY         A I
...
```

- *diccionario de relleno* (*filler dictionary*) o *diccionario de ruido*, que usualmente define las palabras legales que no están en el modelo del lenguaje pero que son parte del lenguaje mismo, como aspiraciones, pausas, etc. Este diccionario debe tener por lo menos los siguientes valores:

```
< S >     SIL
< SIL >   SIL
< /S >    SIL
```

  donde `<S>` es el silencio en el comienzo de la pronunciación de la oración,

< SIL >, el silencio dentro de la pronunciación y </S> el silencio al final de la pronunciación de la oración. Estas palabras son tratadas como palabras especiales y es necesariao que sean representadas en el diccionario de relleno. Por lo menos una de ellas debe estar relacionada con un fonema llamado "SIL", que representa el silencio entre palabras. *Sphinx* espera que se nombren los eventos acústicos para las condiciones generales de ruido fondo como SIL. Otros ruidos pueden ser modelados también por fonemas definidos por el usuario.

- Un *archivo con la lista de fonemas* (*phonelist*) que es una lista de todas las unidades acústicas que se han elegido para entrenar el modelo acústico. *Sphinx* no permite unidades diferentes a las de sus diccionarios. Todas las unidades en sus dos diccionarios deben ser listadas aquí. En otras palabras, su lista de fonemas debe tener exactamente las mismas unidades usadas en sus diccionarios, no más ni tampoco menos. Cada fonema debe ser listado en una línea separada dentro del archivo, comenzando con la izquierda, con ningún espacio extra después del fonema. Por ejemplo:

```
A
A_1
B
CH
D
E
E_1
F
G
⋮
```

Apéndice E

Arquitectura de software de *SphinxTrain*

El módulo *SphinxTrain* consiste de 40 aplicaciones basadas en lenguaje C y de 9 scripts hechos en Perl [6]. A continuación se presentan las distintas categorías en que se dividen de acuerdo al tipo de tareas generales que desempeñan y se detallarán solamente las aplicaciones que se analizaron y utilizaron para el entrenamiento de HMM continuos.

1. **Rutinas de conversión de modelos**. Incluye rutinas para transformar modelos en formato *Sphinx 3* a formato *Sphinx 2*, como `smk_s2cb`, `mk_s2hmm`, `mk_s2phone`, `mk_s2phonemap` y `mk_s2sendump`,y para transformar modelos en formato *Sphinx 2* a formato *Sphinx 3*, como `mk_s3gau`, `mk_s3mixw`, `mk`s3tmat y `mk_ts2cb`. Estas rutinas no se emplean en este trabajo por lo que no se detallarán sus descripciones y parámetros.

2. **Rutinas de manipulación de modelos**. Esta categoría incluye rutinas relacionadas tanto para observar los parámetros de los modelos acústicos obtenidos, como para copiar parámetros del modelo así como para modificar la cantidad de gaussianas que se entrenan en los modelos CD. Se incluyen las siguientes aplicaciones:

   - `printp`, que se encarga de desplegar los parámetros de los modelos acústicos (matrices de transición, mezclas gaussianas, medias y variancias) así como otras estructuras de datos importantes, como el modelo de definición.En la tabla E.2 se muestran los parámetros de la aplicación `printp`.
   - `cp_parm`, que se encarga de copiar los parámetros Gaussianos iniciales (medias y variancias) a los estados del modelo. En la tabla E.3 los parámetros correspondeintes a `cp_parm`.
   - `inc_comp`, que se encarga de incrementar el número de densidades gaussianas del modelo,

- `mk_model_def` y `mk_mdef_gen`, que se encargan de crear el archivo de definición del modelo en distintos formatos. El ejecutable `mk_model_def` prepara el archivo de definición de modelo con los fonemas y los trifonemas. Por su parte `mk_mdef_gen` es una versión mejorada de `mk_model_def`, además de obtener el modelo para los fonemas CI, también genera los trifonemas a partir del diccionario de palabras (todos los trifonemas posibles), el archivo contador de los fonemas CI y los trifonemas que existen en el archivo de transcripción del cuerpo de entrenamiento así como el archivo con los fonemas CI y los trifonemas recortados (trifonemas con una mínima y una máxima ocurrencia). Sus parámetros se encuentran en la tabla E.5. Los parámetros de `mk_model_def` no se incluyen debido a que no se utiliza en este trabajo.

3. **Rutinas de adaptación de modelos**. En donde se incluyen las aplicaciones `mk_mllr_class`, que crea clases de regresión, `mllr_transform`, que transforma un conjunto basado en un modelo en un conjunto de clases de regresión, `map_adapt`, que puede adaptar un conjunto de un modelo que se basa en el criterio de máximo a posteriori (MAP), `mllr_solve`, que calcula la matriz de regresión basada en la estadística suficiente de la adaptación de los datos. Esta categoría no fue utilzada en este trabajo por lo que se omitirán sus parámetros.

4. **Rutina de extracción de características**. Que comprende solamente al ejecutable `wave2feat`, que se encarga de obtener los coeficientes MFCC de una señal de voz. Sus parámetros se muestran en la tabla E.6.

5. **Rutinas de manipulación de árboles de decisión**. Comprende aquellas aplicaciones que permiten crear las preguntas lingüísticas y los árboles de decisión asociados con ellas, así como realizar el podado de tales árboles y el ligado de estados CD. Sus aplicaciones son:

   - `bldtree`, que se encarga de crear un árbol de decisión con base en un conjunto de preguntas lingüísticas. Sus parámetros se describen en la tabla E.8.
   - `tiestate` , que se encarga de ligar los estados basado en las árboles de decisión y cuyos argumentos se detallan en la tabla E.9.
   - `prunetree`, que se encarga de podar un árbol de decisión haciéndolo de un tamaño más razonable para facilitar su procesamiento. En la tabla E.10 se describen sus parámetros.
   - `make_quests`, que genera un conjunto de preguntas lingüísticas en forma automática. La aplicación `make_quests` utiliza los argumentos de la tabla E.7.

6. **Rutinas de inicialización de modelos**. En esta categoría se incluyen a todas aquellas aplicaciones que se encargan de realizar la estimación de

los parámetros inciales de los HMM en varias etapas del entrenamiento. Las aplicaciones de esta categoría son:

- `mk_flat`, que se encarga de realizar la inicialización de los pesos mixtos y de la matriz de transición inicial. El ejecutable `mk_flat` tiene los argumentos de la tabla E.11.
- `init_gau`, que se encarga de la inicialización de la media y la variancia globales de los HMM. El ejecutable `init_gau` tiene los argumentos de la tabla E.12.
- `init_mixw`, que realiza la inicialización de los parámetros del modelo no ligado CD. Los argumentos de `init_mixw` se muestran en la tabla E.13
-

7. **Rutinas de estimación de modelos**. En esta categoría se encuentras las aplicaciones más importantes para el entrenamiento ya que se encargan de la reestimación de los parámetros de los HMM. Se incluyen las aplicaciones:

   - `bw` que se utiliza para llevar a cabo la estimación Baum-Welch y generar las estadísticas suficientes de los modelos CI, no ligados CD, CD ligados con una o más mezclas gaussianas. La descripción de los parámetros de la aplicación `bw` se muestran en las tablas E.15 y E.16.
   - `norm`, que recolecta las estadísticas suficientes y genera los modelos CI, no ligados CD y ligados y de las mezclas gaussianas. El ejecutable `norm` tiene los parámetros que se detallan en la tabla E.14.
   - `delint`, que lleva a cabo la interpolación borrada para HMM semicontinuos,
   - `mixw_interp`, que lleva a cabo la interpolación de los pesos de las mezclas de los HMM semicontinuos.

8. **Rutinas misceláneas**. Comprende aplicaciones como `dict2tri`, que convierte el archivo de transcripción en trifonemas, `param_cnt`, que permite contar los parámetros de varios recursos y `quick_count`, que puede ser usado para generar la lista de todos los posibles trifonemas de un diccionario temporal que contenga todas las palabras del vocabulario excepto las palabras de relleno.

En cuanto a los scripts, el script `make_topology.pl` se utiliza para la creación del archivo de topología, archivo necesario antes de comenzar el entrenamiento. Utiliza los argumentos definidos en la tabla E.1.

Los scripts restantes definen varios módulos o etapas en las que se puede dividir el entrenamiento de HMM continuos y semicontinuos:

1. MODULO 00. Es el módulo de verificación de los archivos de entrenamiento. Realiza las siguientes tareas:

Argumento	Default	Descripción
Estados por hmm	–	Número de estados por HMM
`skipstate`	no	"yes" o "no" dependiendo si desea que el HMM tenga saltos en las transiciones de estados.

Tabla E.1: Parámetros del script `make_topology.pl`

- Se asegura de que los diccionarios, tanto el de pronunciación como el de sonidos de relleno, sean congruentes con la lista de las unidades acústicas.
- Verifica que no haya entradas repetidas en el diccionario.
- Verifica que existan todos los archivos que se listan en el archivo de control.
- Verifica que el número de líneas en el archivo de transcripción coincidan con todos los archivos que se listan en el archivo de control.
- Determina si la cantidad de datos de entrada es suficiente o razonable para empezar el entrenamiento.
- Revisa que todas las palabras en la trascripción aparezcan en el diccionario.
- Verifica si todos los fonos en el archivo de transcripción está en la lista de unidades acústicas y si todos los fonos de las unidades acústicas aparecen al menos una vez.

El script que corre este módulo es: `scripts_pl/00.verify/verify_all.pl`

2. MODULO 10. Es el módulo donde se realiza la cuantificación de vectores que es una técnica de codificación que ha sido aplicada con éxito tanto a compresión de habla como de imágenes. Su tarea consiste en agregar los vectores de características a un solo archivo para después, realizar un cómputo de los centroides en el espacio de vectores. Sin embargo, no aplica en el caso de HMM continuos. El script que corre este módulo es: `scripts_pl/10.vector_quantize/slave.VQ.pl`

3. MODULO 20. Este módulo realiza el entrenamiento de los modelos independientes del contexto. Aquí se realizan la inicialización plana de los parámetros CI, las iteraciones con el algoritmo Baum-Welch y la normalización de los parámetros CI y se copian dichos parámetros a todos los estados HMM. El script que corre este módulo es: `scripts_pl/20.ci_hmm/slave_convg.pl`

4. MODULO 30. Es el módulo donde se entrenan los modelos dependientes del contexto no ligados. En él se crean los trifonemas, se crea el archivo de definición para los modelos no ligados CD, se realiza la inicialización de los parámetros no ligados CD se copian dichos parámetros a todos los estados HMM. El script que corre este módulo es: `scripts_pl/30.cd_hmm_untied/slave_convg.pl`

5. MODULO 40. En este módulo se construyen los árboles de decisión para compartir parámetros y los modelos dependientes del contexto ligados. En él se lleva a cabo la generación de las preguntas lingüísticas y de los árboles de decisión para cada estado. También en este módulo se realiza el podado de los árboles de decisión y la obtención de los modelos CD ligados. El script que corre este módulo es: `scripts_pl/40.buildtrees/slave.treebuilder.pl`

6. MODULO 50 En este módulo se reentrenan los modelos dependientes del contexto ligados con una o más Gaussianas. Aquí se realiza la estimación Baum-Welch empezando con una Gaussiana, después se incrementa el número de Gaussianas en potencias de 2 hasta un número máximo de Gaussianas que se haya establecido. Cada vez que se generan Gaussianas se reestiman los parámetros de los modelos CD ligados. El script que corre este módulo es: `scripts_pl/50.cd_hmm_tied/slave_convg.pl`

7. MODULO 90. Es el módulo donde se realiza el borrado de interpolaciones. No aplica en el caso de modelos continuos. El script que corre este módulo es: `scripts_pl/90.deleted_interpolation/deleted_interpolation.pl`

8. MODULO 99. En éste módulo se realiza la conversión de los modelos al formato de *Sphinx 2*. No es necesario en el caso de modelos para *Sphinx 3* . El script que corre este módulo es: `scripts_pl/99.make_s2_models/make_s2_models.pl`

Cabe destacar que para hacer uso de estos módulos, *Sphinx* cuenta con dos archivos de configuración, uno para configurar los parámetros del entrenamiento y otro para configurar los parámetros del reconocimiento, llamados `sphinx_train.cfg` y `sphinx_decode.cfg`, respectivamente. Entre otras cosas contienen todos los directorios y parámetros por omisión que maneja *Sphinx* en las aplicaciones de los distintos módulos. No se tratarán en este apartado debido a que la configuración de las aplicaciones se hizo en forma manual.

Bandera	Default	Descripción
`tmatfn`	–	Muestra el contenido del archivo de matrices de transición.
`mixwfn`	–	Muestra el contenido del archivo de los pesos de las mezclas.
`mixws`	–	Si se desea mostrar sólo un rango del archivo de los pesos de las mezclas se establece el identificador de inicio del intervalo.
`mixwe`	–	Si se desea mostrar sólo un rango del archivo de los pesos de las mezclas se establece el identificador de fin del intervalo.
`gaufn`	–	Muestra el contenido de un archivo de parámetros Gaussianos. Puede ser el de las medias o el de las variancias
`fullgaufn`	–	Muestra el contendido de un archivo que tenga matrices de covariancias completas (no diagonales).
`gaucntfn`	–	Muestra el contenido de un archivo de vectores de pesos de parámetros Gaussianos.
`regmatcntfn`	–	Muestra el contenido de un archivo de conteo de matrices de regresión obtenidas por MLLR
`ldafn`	–	Muestra el contenido de un archivo de transformación por LDA.
`moddeffn`	–	Muestra el archivo de definición del HMM.
`lambdafn`	–	Muestra el archivo de pesos interpolados.
`lambdamin`	0	Muestra los pesos de interpolación mayores o iguales al indicado por esta bandera.
`lambdamax`	1	Muestra los pesos de interpolación menores o iguales al indicado por esta bandera.

Tabla E.2: Parámetros de `printp`.

Bandera	Default	Descripción
`cpopsfn`	–	Nombre del archivo de mapa de operaciones de copia.
`imixwfn`	–	Nombre del archivo de pesos mixtos de estrada (Input mixing weight file).
`omixwfn`	–	Archivo de pesos mixtos de salida.
`nmixwout`	–	Número de arreglos de pesos mixtos en el archivo de salida.
`itmatfn`	–	Archivo de entrada de la matriz de transición.
`otmatfn`	–	Archivo de salida de la matriz de transición.
`ntmatout`	–	Número de matrices de transición en el archivo de salida.
`igaufn`	–	Archivo de entrada de los parámetros de la densidad Gaussiana. Es decir, el archivo de entrada con la media (o variancia) inicializada.
`ogaufn`	–	Archivo de salida de los parámetros de la densidad Gaussiana. Esto es, el archivo de salida con la media (o variancia) inicializada.
`ncbout`	–	Número de codebooks en el archivo de salida. Número de estados por HMM por cada fonema. (ej, total del número de estados).
`feat`	$c/1..L-1/$, $d/1..L-1/$, $c/0/d/0/dd/0/$, $dd/1..L-1/$	Flujo de características a usar. Véase el apéndice C.

Tabla E.3: Parámetros de `cp_parm`.

Bandera	**Default**	**Descripción**
`ninc`	1	Indica el número de Gaussianas (por estado) a dividir. Generalmente potencia de 2. No se necesita aumentar al doble el número de Gaussianas, se pueden especificar otros números, tan grandes como el número de Gaussianas que se tiene actualmente. Este es un número entero positivo. El número de Gaussianas resultante será: $n\_densidades\_actuales + ninc$.
`ceplen`	13	Tamaño del vector de características base.
`dcountfn`	–	Archivo de entrada de los pesos mixtos.
`inmixwfn`	–	Archivo de entrada de los pesos mixtos para todas las den/mix.
`outmixwfn`	–	Archivo de salida de los pesos mixtos.
`itmatfn`	–	Archivo de entrada de la matriz de transición.
`otmatfn`	–	Archivo de salida de la matriz de transición.
`inmeanfn`	–	Archivo de entrada de las medias.
`outmeanfn`	–	Archivo de salida de las medias.
`invarfn`	–	Archivo de entrada de las variancias.
`outvarfn`	–	Archivo de salida de las variancias.
`feat`	–	Flujo de características a usar. Véase el apéndice C

Tabla E.4: Parámetros de `inc_comp`.

Bandera	Default	Descripción
`phnlstfn`	-	Nombre del archivo con la lista de los fonemas .
`inCImdef`	-	Nombre del archivo de definición del modelo CI que se usa como entrada (si de ha dado el argumento `phnlstfn` se ignora su valor).
`triphnlstfn`	-	Nombre del archivo de trifonemas para *Sphinx 3* (si se han dado los argumentos `phnlstfn` ó `inCImdef` se ignora su valor).
`inCDmdef`	-	Nombre del archivo de definición del modelo CD que se usa como entrada (si se han dado los argumentos `phnlstfn`, `inCImdef` ó `triphnlstfn` se ignora su valor).
`dictfn`	-	Nombre del archivo de diccionario de palabras.
`fdictfn`	-	Nombre del archivo de diccionario de relleno.
`lsnfn`	-	Nombre del archivo de transcripción.
`n_state_pm`	3	Número de estados por HMM en los modelos que se desean entrenar.
`ocountfn`	-	Nombre del archivo de salida del conteo de fonemas y de trifonemas.
`ocimdef`	-	Nombre del archivo de salida para la definición del modelo CI.
`oalltphnmdef`	-	Nombre del archivo de salida para la definición del modelo de todos los trifonemas .
`ountiedmdef`	-	Nombre del archivo de salida para la definición del modelo no ligado.
`minocc`	1	Número mínimo de ocurrencias de un trifonema para ser contemplado o incluido en el archivo de definición del modelo (mdef).
`maxtriphone`	100000	Numero máximo de trifonemas deseado en el archivo mdef.

Tabla E.5: Parámetros de `mk_mdef_gen`

Bandera	Default	Descripción
`srate`	–	Frecuencia de muestreo de la señal de entrada.
`frate`	100	Indica que tan rápido se mueve la ventana. Se hace en´términos del número de muestras.
`wlen`	0.025625	El tamaño de la ventana de Hamming en segundos.
`nfft`	512	Tamaño de la FFT.
`nfilt`	40	Numero de filtros utilizado en el banco de filtros.
`ncep`	13	Numero de coeficientes cesptral incluyendo el coeficiente de energía en el vector de características.
`alpha`	0.97	Valor del coeficiente de preéfasis (α).
`lowerf`	133.33334	Frecuencia más baja del filtro de menor orden en el banco de filtros.
`upperf`	6855.4976	Frecuencia más alta del filtro de mayor orden en el banco de filtros.
`wlen`	0.025625	El tamaño de la ventana de Hamming en segundos.
`i`	–	Nombre de un sólo archivo de audio que se usa como entrada.
`o`	–	Nombre de un sólo archivo de audio que se usa como salida.
`c`	–	Archivo de control para procesamiento en modo batch.
`nskip`	–	Si se especifica un archivo de control, indica el número de locuciones que se deben de saltar a partir del inicio del archivo.
`runlen`	–	Si se especifica un archivo de control, indica el número de locuciones que se deben de procesar a partir del inicio del archivo. (véase también `nskip`).
`di`	–	Directorio de entrada. Si es definido, los nombres de los archivos de entrada son relativos a éste
`ei`	–	Extensión que se añade a los archivos de entrada.
`do`	–	Directorio de salida. Si es definido, los nombres de los archivos de salida son relativos a éste.
`eo`	–	Extensión que se añade a los archivos de salida.
`nist`	no	Define el formato de entrada de los datos como NIST.
`raw`	no	Define el formato de entrado de los datos como datos brutos en binario (raw).
`mswav`	no	Define el formato de entrada de los datos como Microsoft Wav.
`input_endian`	little	Indica si los datos de entrada se encuentran en big endian o little endian. Se ignora con las banderas `nist` y `mswav`.
`nchans`	1	Número de canales que se manejan. Se asumen muestras entrelazadas.
`whichchan`	1	Número de canal a procesar.
`logspec`	no	Genera archivos logcepstral en lugar de cepstral.
`feat`	sphinx	Indica que los archivos de entrada están en el formato que maneja *Sphinx* (big endian).
`mach_endian`	little	Indica si la máquina trabaja con big endian o con little endian.
`doublebw`	no	Utiliza filtros de doble ancho de banda. Equivale a tomar una frecuencia central.
`warp_type`	`inverse_linear`	Tipo de función de ajuste o contorno.
`warp_params`	–	Parámetros que definen la función de ajuste.
`blocksize`	200000	Tamaño del bloque. Limita el número máximo de muestras utilizadas en el tiempo cuando se leen archivos de audio de gran duración.
`dither`	no	Añade ruido de 1/2−bit. Esta opción es útil para la decodificación de la señal debido a que las tramas que contengan solamente ceros podrían causar que el frontend caiga en una condición de división entre cero.
`seed`	−1	Semilla del generador de ruido aleatorio.
`verbose`	no	Muestra los archivos de entrada.

Tabla E.6: Parámetros de `wave2feat`.

Bandera	Default	Descripción
`moddeffn`	–	Archivo de definición del modelo CI.
`meanfn`	–	Archivo de las medias CI.
`varfn`	–	Nombre del archivo de las variancias CI.
`mixwfn`	–	Archivo de los pesos mixtos CI.
`npermute`	6	Un algoritmo de agrupamiento (clustering) abajo-arriba (bottom-up) y de arriba-abajo (top-down), escrito por Rita Singh, se usa para agrupar los fonemas en clases. Los fonemas son agrupados usando el agrupamiento abajo-arriba hasta que se obtengan `npermute` clases. Las `npermute` clases son particionadas en forma exhaustiva en dos clases y evaluadas para identificar la partición óptima del conjunto entero de fonemas en dos grupos. Para generar un árbol completo, se lleva a cabo un proceso idéntico, que es ejecutado recursivamente en cada uno de esos dos grupos. El valor de `npermute` típicamente se encuentra entre 8 y 12 . Valores pequeños en `npermute` resultan en un agrupamiento sub-óptimo. Valores grandes resultan computacionalmente prohibidos.
`niter`	0	El agrupamiento abajo-arriba (bottom-up) y de arriba-abajo (top-down) puede ser iterado para obtener grupos que sean más óptimos. `niter` establece el número de iteraciones a ejecutar. El valor típico de `niter` es 1 ó 2, puesto que el agrupamiento se satura después de 2 iteraciones.
`qstperstt`	8	El algoritmo agrupa distribuciones de estados correspondiente a cada estado de los HMM de los fonemas CI para generar las preguntas lingüísticas. De esta forma, todos los primeros estados son agrupados para generar un subconjunto de preguntas y los segundos estados son agrupados para generar un segundo subconjunto y así sucesivamente. El valor dado a `qstperstt` determina cuantas preguntas son generadas para agrupar cualquier estado. Típicamente este valor se establece a un número entre 20 y 25.
`varfloor`	$1.0e-08$	Valor mínimo de las variancias.
`questfn`	–	Archivo de salida de las preguntas lingüísticas.
`fullvar`	no	El archivo de variancias contiene matrices de covariancia completas (no diagonales).

Tabla E.7: Parámetros de `make_quests`.

Bandera	Default	Descripción
`treefn`	–	Nombre del archivo de salida para almacenar los árboles producidos. Se debe proporcionar la ruta completa.
`moddeffn`	–	Archivo de definición del modelo no ligado CD.
`ts2cbfn`	`.semi.`	Tipo de modelos que se esta entrenando. Para el caso continuo se debe escribir .cont., para el caso semicontinuo .semi.
`mixwfn`	–	Archivo de pesos mixtos del modelo no ligado CD.
`meanfn`	–	Archivo de las medias del modelo no ligado CD.
`varfn`	–	Archivo de las variancias del modelo no ligado CD.
`varfloor`	0.00001	Valor mínimo de la variancia.
`mwfloor`	$1e-4$	Valor mínimo de los pesos mixtos. Cantidades más pequeñas a este valor, son cambiadas a este mínimo valor. Un valor típico podría ser $1e-8$.
`psetfn`	–	Archivo de preguntas lingüísticas.
`phone`	–	Fonema CI para el que desea construir el árbol de decisión.
`state`	–	Posición del estado HMM para el cual se desea construir el árbol de decisión. Para un modelo de 3 estados HMM, este valor puede ser 0, 1 ó 2. Para un modelo de 5 estados HMM, este valor puede ser 0, 1, 2, 3, ó 4, y así sucesivamente para el modelo de n estados HMM.
`stwt`	–	Pesos de los estados vecinos. Esta bandera necesita una cadena de números igual al número de estados HMM, por ejemplo, si se utilizan 5 estados HMM, entonces los argumentos de esta bandera pueden ser `-stwt` 1.00.30.10.010.001. Cada uno de esos números especifica el peso dado a las distribuciones del estado, cuando se construye del árbol, comenzando con el estado actual. El segundo número especifica el peso dado a los estados inmediatamente adyacentes al estado actual (si es que existe alguno), el tercer número especifica el peso dado a estados adyacentes que son removidos de aquellos estados inmediatos adyacentes (si es que existe alguno), y así sucesivamente. Un conjunto típico de valores para modelos de 5 estados HMM es 1.00.30.10.010.001.
`ssplitmin`	1	Las preguntas complejas son construidas para el árbol de decisión. En primer lugar se construye un "pre-árbol" usando las preguntas lingüísticas en el archivo de preguntas. El número mínimo de bifurcaciones en este árbol es dado por `ssplitmin`. Este valor no debe ser mas bajo que 1.
`ssplitmax`	5	El número máximo de bifurcaciones en el árbol simple, antes que éste sea usado para construir las preguntas complejas. Este número se establece típicamente a 7. Valores más grandes serán más intensivos computacionalmente. El valor dado no debe ser menor a `ssplitmin`.
`ssplitthr`	$8e-4$	Aumento mínimo en la probabilidad para ser considerado en una bifurcación en el árbol simple. Típicamente se establece a un número muy pequeño, más grande ó igual a 0.
`csplitmin`	1	El número mínimo de bifurcaciones en el árbol de decisión. Este valor no debe ser menor a 1.
`csplitmax`	100	El máximo número de bifurcaciones en el árbol de decisión. Éste debe ser computacionalmente tan grande como sea posible . Un valor típico puede ser 2000.
`csplitthr`	$8e-4$	El incremento mínimo en la probabilidad a ser considerada para una bifurcación en el árbol de decisión. Típicamente se establecen valores muy pequeños, mayores ó iguales a 0.
`cntthresh`	0.00001	Número mínimo de observaciones en un estado para que éste sea considerado en el proceso de construcción del árbol de decisión. Se ignoran todos los estados con el número de observaciones menores a este valor.

Tabla E.8: Parámetros de `bldtree`.

Bandera	Default	Descripción
`imoddeffn`	–	Archivo de definición del modelo con todos los trifonemas.
`omoddeffn`	–	Archivo de definición del modelo ligado CD.
`treedir`	–	Directorio donde se almacenan los archivos de árboles de decisión recortados o podados.
`psetfn`	–	Archivo de preguntas lingüísticas.

Tabla E.9: Parámetros de la aplicación `tiestate`.

Bandera	Default	Descripción
`itreedir`	–	Directorio en el que se encuentran almacenados todos los árboles de decisión.
`nseno`	–	Número de senones que se desea entrenar.
`otreedir`	–	Nombre del directorio para almacenar los árboles de decisión podados.
`moddeffn`	–	Archivo de definición del modelo no ligado CD (CD untied file) .
`psetfn`	–	Archivo de preguntas lingüísticas.
`minocc`	0.0	Número mínimo de observaciones en el estado ligado dado. Si hay pocas observaciones, las ramas correspondientes al estado ligado se recortan por defecto. Este valor nunca debe ser cero, de lo contrario se obtendrán senones con ninguna información para entrenar, puesto que son observados cero veces en el conjunto de entrenamiento.

Tabla E.10: Parámetros de `prunetree`.

Bandera	Default	Descripción
`moddeffn`	–	El nombre del archivo de definición del modelo para *Sphinx 3* (vea `mk_model_def`)
`mixwfn`	–	Nombre del archivo en donde se almacenará los pesos mixtos (`mixture_weights`) para *Sphinx 3*
`topo`	–	Un archivo de plantilla para la matriz de topología del HMM (vea el script `make_topology.pl`)
`tmatfn`	–	Nombre del archivo donde desea escribir las matrices de transición inicializadas (`transition_matrix`)
`nstream`	4	Numero de flujos de características independientes, para modelos continuos este debe de establecerse a 1.
`ndensity`	256	Numero de densidades gaussianas por cada estado. Para modelos CI este número debe establecerse a 1.

Tabla E.11: Parámetros de la aplicación `mk_flat`.

Bandera	Default	Descripción
`moddeffn`	–	El nombre del archivo de definición del modelo para *Sphinx 3* (vea `mk_model_def`)
`ts2cbfn`	–	Nombre del archivo para establecer relaciones de estados ligados a codebook (tied-state-tocodebook). Esta bandera debe ser establecida a ".cont" si se está entrenando modelos continuos, y a ".semi" si se está entrenando con modelos semicontinuos, sin las dobles comillas.
`accumdir`	–	Directorio donde se desea escribir el buffer intermedio
`meanfn`	–	Nombre del archivo en el cual desea escribir la media global.
`ctlfn`	–	Archivo de control de cuerpo de entrenamiento
`nskip`	–	Si se especifica un archivo de control, indica el número de locuciones que se deben de saltar a partir del inicio del archivo.
`runlen`	–	Si se especifica un archivo de control, indica el número de locuciones que se deben de procesar a partir del inicio del archivo. (véase también `nskip`).
`part`	–	Numero de la parte. Identifica el numero de parte del cuerpo (rango de $1..NPART$)
`npart`	–	Total de número de partes.
`lsnfn`	–	- Nombre del archivo de trascripción.
`dictfn`	–	Archivo que contiene el diccionario de palabras.
`fdictfn`	–	Diccionario de palabras de relleno .
`segdir`	–	- Ruta del directorio raíz del archivo de segmentación de estados en el cuerpo de entrenamiento.
`scaleseg`	no	Escala de segmentación existente para ajustar el nuevo parámetro de longitud del flujo.
`segext`	v8_seg	Extensión de los archivos de segmentación de estados en el cuerpo de entrenamiento.
`cepdir`	–	Directorio raíz de los archivos cepstral del cuerpo de entrenamiento. Esta dirección se agregará a todas las rutas en el archivo de control.
`cepext`	mfc	Extensión de los archivos cepstral del cuerpo de entrenamiento.
`silcomp`	none	Realizar la compresión de silencio. Sus posibles valores son current, sildelfn, none. Con current elimina las tramas que tienen energía menor a un umbral calculado mediante un porcentaje de la energía normalizada. Con sildelfn se elimina un número de de tramas por dos arreglos, uno para el comienzo de la eliminación y el otro para el fin. none no elimina silencios
`feat`	–	Tipo de características. Control para definir el algoritmo de extracción de características que debe usarse. Debe de ser cualquiera de los valores indicados en el apéndice C.
`ceplen`	13	Dimensión de los vectores cepstral
`agc`	max	Control automático de ganancia (noise, max, emax, none).
`cmn`	current	Control de normalización de la media, los valores posibles son: (current, prior y none).
`varnorm`	no	Control de normalización de la variancia (yes/no).

Tabla E.12: Parámetros de la aplicación `init_gau`

Bandera	**Default**	**Descripción**
`src_moddeffn`	–	Archivo fuente de definición del modelo CI.
`src_ts2cbfn`	–	Tipo de entrenamiento del archivo fuente: continuo (.cont.) o semicontinuo (.semi.).
`src_mixwfn`	–	Archivo fuente de los pesos mixtos CI
`src_meanfn`	–	Archivo fuente de las medias CI.
`src_varfn`	–	Archivo fuente de las variancias CI
`src_tmatfn`	–	Archivo fuente de las matrices de transición CI.
`dest_moddeffn`	–	Archivo destino de definición del modelo CD.
`dest_ts2cbfn`	–	Tipo de entrenamiento del archivo de destino: continuo (.count.) o semicontinuo (.semi.).
`dest_mixwfn`	–	Archivo destino de los pesos mixtos CD.
`dest_meanfn`	–	Archivo destino de las medias CD.
`dest_varfn`	–	Archivo destino de las variancias CD.
`dest_tmatfn`	–	Archivo destino de las matrices de transición CD.
`feat`	$1s\_12c\_12d\_3p\_12dd$	Tipo de flujo de características. Véase el apéndice C.
`fullvar`	no	Tanto el archivo de origen y el de destino contienen matrices de covariancia completas (no diagonales).
`ceplen`	13	Dimensión de los vectores cepstral.

Tabla E.13: Parámetros de `init_mixw`.

Bandera	Default	Descripción
`accumdir`	-	Directorios de los buffers donde se encuentra la suma para la media.
`meanfn`	-	Archivo en el cual desea guardar la media global. Archivo de la media de la densidad gaussiana (si existe).
`feat`	-	Tipo de flujos de características. Véase el apéndice C.
`ceplen`	-	Dimensión de los vectores característicos base.
`oaccumdir`	-	Ruta de la carpeta que contiene todas las sumas restimadas.
`tmatfn`	-	Archivo de la matriz de probabilidades de transición para producir (si existe).
`mixwfn`	-	Archivo de pesos mixtos para producir (si existe).
`varfn`	-	Archivo en donde se desea almacenar la variancia de la densidad gaussiana (si existe).
`regmatfn`	-	Archivo en donde se almacenan las matrices de regresión MLLR (si existe).
`dcountfn`	-	Archivo a crearse para la contar las densidades gaussianas.
`inmixwfn`	-	Usa los pesos mixtos de este archivo si nunca se observan.
`inmeanfn`	-	Usa la media de este archivo si nunca se observa.
`invarfn`	-	Usa la variancia de este archivo si nunca se observa
`ifullvar`	no	Las variancias están dadas como matrices de covariancia completas (no diagonales).
`tiedvar`	no	Liga todas las matrices de covariancia.

Tabla E.14: Parámetros de la aplicación `norm`.

Bandera	Default	Descripción
`moddeffn`	–	Archivo de definición del modelo para el inventario del modelo a entrenar (CI, no ligados CD, CD ligados).
`ts2cbfn`	–	Nombre del archivo para para establecer relaciones de estados ligados a codebook (tied-state-tocodebook). Esta bandera debe ser establecida a ".cont" si se está entrenando modelos continuos, y a ".semi" si se está entrenando con modelos semicontinuos, sin las dobles comillas.
`mixwfn`	–	Nombre del archivo en donde los pesos mixtos son almacenados de la iteración previa. Se debe proporcionar la ruta absoluta.
`mwfloor`	0.00001	Valor mínimo de los pesos mixtos. Cualquier número por debajo de este valor mínimo se establece a este valor.
`tmatfn`	–	Nombre del archivo en cuál las matrices de transición son almacenadas de la iteración previa. Se debe proporcionar la ruta absoluta.
`tpfloor`	0.0001	Valor mínimo de las probabilidades de transición. Cualquier número debajo del este valor mínimo se establece a este valor.
`meanfn`	–	Nombre del archivo en el que las medias son almacenadas en la iteración previa. Se debe proporcionar la ruta absoluta.
`varfn`	–	Nombre del archivo donde se almacenaron las variancias en la iteración previa. Se debe proporcionar la ruta absoluta.
`dictfn`	–	Archivo que contiene el diccionario de palabras.
`fdictfn`	–	Diccionario de palabras de relleno .
`ctlfn`	–	Archivo de control del cuerpo del entrenamiento.
`nskip`	–	Si se especifica un archivo de control, indica el número de locuciones que se deben de saltar a partir del inicio del archivo.
`runlen`	–	Si se especifica un archivo de control, indica el número de locuciones que se deben de procesar a partir del inicio del archivo. (véase también `nskip`).
`part`	–	El corpus de entrenamiento se puede dividir en N partes iguales con solo establecer la bandera `part`. Si hay M pronunciaciones en su archivo de control, entonces se podrá correr el entrenamiento separadamente en cada (M/N) partes. Esta bandera puede ser establecida para especificar cual de esas partes desea entrenar actualmente. Por ejemplo, sí el número total de partes es 3, esta bandera puede tomar uno de los valores $1, 2$ ó 3.
`npart`	–	Número de partes en el cual desea dividir el corpus de entrenamiento.
`cepdir`	–	Directorio donde se almacenan los archivos de características.
`cepext`	mfc	La extensión que viene después del nombre listado en el archivo de control. Por ejemplo, se puede tener un archivo llamado $a/b/c.d$ y en el archivo de controlla puede tener la línea $a/b/c$. Entonces esta bandera debe darse como argumento "d", sin las dobles comillas o el punto antes de ésta.
`segdir`	–	Ruta del directorio raíz del archivo de segmentación de estados.
`sentdir`	–	Ruta del directorio del archivo de transcripción.
`sentext`	sent	La extensión del archivo de transcripción.
`lsnfn`	–	Nombre del archivo de trascripción.
`accumdir`	–	Directorio donde se almacenarán los resultados intermedios de cada parte del entrenamiento. Si tiene T medias para estimar, entonces el tamaño del buffer de la media de la parte actual del entrenamiento, será $T * 4$ bytes. De igual forma serán los buffers para la variancia, para los pesos mixtos y para las matrices de transición.
`varfloor`	0.00001	Valor mínimo permitido de la varianxia.
`topn`	4	Número de Gaussianas a considerar para el cálculo de la probabilidad de cada estado. Por ejemplo, si tiene 8 modelos de *gaussianas/estado* y `topn` es 4, entonces las 4 gaussianas más probables son usadas.
`abeam`	$1e-100$	Evalua los valores alfa del algoritmo hacia adelante con respecto al ancho de este haz (beam) (Forward Beamwidth).
`bbeam`	$1e-100$	Evalúa los valores beta del algoritmo hacia atrás (Actualización de las sumas de restimación) con respecto al ancho de este haz (beam) (Backward Beamwidth).
`agc`	max	El tipo de control automático de la ganancia que se realizará (max o emax).
`cmn`	current	Normalización de la media de los cepstral basado en la pronunciación actual o previa (current o prior).

Tabla E.15: Parámetros de `bw`.

Bandera	Default	Descripción
`varnorm`	no	Indica si se realiza la normalización de la variancia (yes/no)
`silcomp`	none	Realizar la compresión de silencio basado en la pronunciación actual o previa (current o prior).
`meanreest`	yes	Reestimación de la media.
`varreest`	yes	Reestimación de la variancia.
`mixwreest`	yes	Reestimación de los pesos mixtos.
`tmatreest`	yes	Reestimación de las matrices de transición.
`timing`	yes	Determina si la información de tiempo es desplegada
`viterbi`	no	Determina si el entrenamiento esta efectuado
`2passvar`	no	Estableciendo esta bandera a "yes" permite que `bw` use las medias previas en la estimación de la variancia. La variancia actual se estima como $E[(x - media\_previa)^2]$. Si esta bandera es establecida a "no" la estimación actual de las medias son usadas para estimar las variancias. Esto necesita que la estimación de la variancia se realice como $E\{x^2\} - E\{x\}^2$, un estimador inestable que algunas veces resulta en estimaciones negativas de la variancia debido a la precisión aritmética.
`sildelfn`	–	Archivo que especifica las tramas de silencio de fondo a borrar.
`spthresh`	0.0	La mínima probabilidad posterior del estado para re-estimar. Los estados abajo de este valor no son contados.
`maxuttlen`	0	Máximo número de tramas para una pronunciación (0 significa que no hay límite fijo).
`ckptintv`	–	Punto de control de suma de reestimación de cada `ckptintv` pronunciaciones. Realiza un almacenamiento de la acumulación para los parámetros de estimación cadas n pronunciaciones, donde n es el valor dado a este argumento.
`ceplen`	13	Longitud de los vectores de características básicos
`feat`	–	Configuración de las vectores de características. Véase el apéndice C.

Tabla E.16: Parámetros de `bw` (cont.).

Apéndice F

Arquitectura de software de *Sphinx Decoder*

La arquitectura de software del decodificador de *Sphinx* se presenta en la figura F.1.

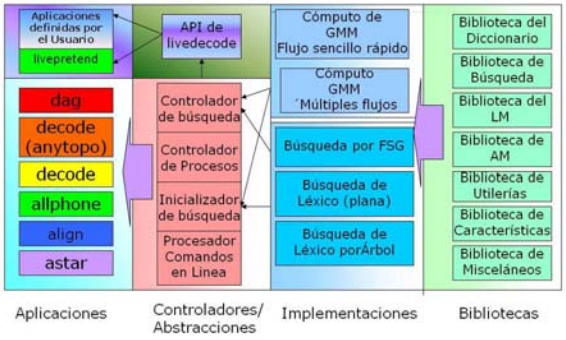


Figura F.1: Arquitectura de software del decodificador de *Sphinx* .

Bandera	Default	Descripción
`ctrlfile`	–	Archivo de control que contiene la lista de los nombres de todos los audios que se van a usar durante la prueba.
`rawdir`	–	Directorio donde se encuentran todas las grabaciones para la prueba que deben ser convertidas a formato raw.
`cfgfile`	–	archivo de configuración del decodificador. Contiene las banderas necesarias para que se efectúe el reconocimiento. Las banderas se detallan en las tablas F.2 y F.3.

Tabla F.1: Parámetros de la aplicación `sphinx3_livepretend`.

Bandera	Default	Descripción
`mdef`	–	Archivo de entrada de definición del modelo .
`mean`	–	Archivo de entrada de las medias de las mezclas Gaussianas.
`var`	–	Archivo de entrada de las variancias de las mezclas Gaussianas.
`mixw`	–	Archivo de entrada de los pesos mixtos de los senones
`tmat`	–	Archivo de entrada de las matrices de transición.
`subvq`	–	Modelo acústico en la forma de subvector cuantizado
`hmm`	–	Archivos del modelo acústico. Se podría especificar el modelo acústico solamente con la opción `hmm`. Los valores por omisión para los componentes HMM son `means`, `variances`, `mixture_weights`,`transition_matrices`y `mdef`(la definición del modelo).
`dict`	–	Archivo que contiene el diccionario de palabras.
`fdict`	–	Diccionario de palabras de relleno .
`lm`	–	Archivo de entrada del modelo del lenguaje basado en trigramas. Se podría especificar el modelo del lenguaje ya sea mediante un archivo binario de descarga (binary dump file) o por medio de un archivo de texto usando `lm`.
`lmctlfn`	–	Se podría especificar un conjunto de clases basadas en el modelo del lenguaje con la opción `lmctlfn`.
`lminmemory`	0	Por omisión, el modelo del lenguaje se accede principalmente desde el disco duro mediante un mecanismo de caché. Se podría habilitar que se activara este comportamiento en el modo de memoria completa especificando la opción `lminmemory`.
`lmname`	–	Nombre del modelo del lenguaje en la opción `lmctlfn` para ser utilizado por todas las pronunciaciones.
`fillpen`	–	Archivo de entrada de las probabilidades de relleno. Se utiliza en lugar de `silpen` y de `noisepen`
`fillprob`	0.1	Valor por omisión de la probabilidad de una palabra de relleno.
`silprob`	0.1	Valor por omisión de la probabilidad del silencio.
`hypseg`	–	Archivo de hipótesis de salida con puntajes y segmentaciones detalladas.
`hypsegfmt`	0	Especifica el formato de segmentación. 0:*Sphinx 3* , 1: *Sphinx 2* o 2:NIST CTM
`hypsegscore_unscale`	1	Cuando se despliegan los resultados, elegir si se quita el escalamiento hacia atrás del puntaje acústico con la mejor ruta en una trama.
`hyp`	–	Archivo con los resultados del reconocimiento que sólo contiene palabras.
`cmn`	current	Normalización de la media de los cepstral basado en la pronunciación actual o previa (current o prior).
`agc`	max	El tipo de control automático de la ganancia para c_0 que se realizará (max o none).
`varnorm`	no	Indica si se realiza la normalización de la variancia (yes/no)
`lowerf`	133.33334	Frecuencia más baja del filtro de menor orden en el banco de filtros.
`upperf`	6855.4976	Frecuencia más alta del filtro de mayor orden en el banco de filtros.
`nfilt`	40	Numero de filtros utilizado en el banco de filtros.
`samprate`	16000.0	Frecuencia de muestreo de la señal de entrada. Solamente toma valores 8000 y 16000 Hz.
`ctlfn`	–	Archivo de control de las pronunciaciones a ser procesadas.
`ctl_lm`	–	(No se usa en los modos 2 y 3) Archivo de control que lista los correspondientes LMS en cada pronunciación.
`ctl_mllr`	–	Archivo de control que lista las correpondientes matriz MLLR para una pronunciación.
`mllr`	sent	Matriz de transformación MLLR que será aplicada a las medias de las mezclas Gaussianas.
`cb2mllr`	`.1cls.`	Archivo de la relación de la matriz de transformación de senón a MLLR.
`bestpath`	0	Decide si recorre o no la mejor ruta en la búsqueda sobre el DAG después de llevar a cabo el algoritmo de Viterbi pass hacia adelante. Es aplicable para los modos de búsqueda 3 (léxico plano) y 4 (árbol de léxico). En el segundo paso controla la búsqueda por mejor ruta.

Tabla F.2: Parámetros de configuración de `sphinx3_livepretend`.

Bandera	Default	Descripción
`bestpathlw`	–	Peso del lenguaje para la mejor ruta en el DAG. Por omisión es el mismo valor que `lw`. Es aplicable para los modos de búsqueda 3 y 4. En el segundo paso controla la búsqueda por mejor ruta.
`ctloffset`	0	Número de pronunciaciones que serán saltadas desde el inicio del archivo de control.
`ctlcount`	1000000000	Número de pronunciaciones que serán procesadas después de haber saltado `ctloffset` entradas.
`outlatdir`	–	Directorio en el cuál se colocarán las celosías de palabras.
`latext`	*lat.gz*	Extensión de los archivos donde se colocarán las celosías de palabras. Por omisión es archivo de tipo comprimido `gzip`.
`beam`	$1.0e-55$	Determina que HMMs permanecen activos en cualquier punto o trama durante el reconocimiento. Se basa en el método de mejor puntaje obtenido en cada estado de cada HMM. Su rango ocupa [`0`(`másamplio`)..`1`(`másestrecho`)
`pbeam`	$1.0e-50$	Determina que HMMs activo puede hacer transiciones hacia sus sucesores en el árbol léxico en cualquier trama. Se basa en el puntaje del estado de salida de un HMM de origen. Su rango ocupa [`0`(`másamplio`)..`1`(`másestrecho`)
`wbeam`	$1.0e-35$	Determina que palabras se reconocen en cualquier trama durante la decodificación. Se basa en los puntajes del estado de salida de un nodo hoja de los HMMs en el árbol de léxico. Su rango ocupa [`0`(`másamplio`)..`1`(`másestrecho`)
`subvqbeam`	$3.0e-30$	Para cada senón y su modelo acústico subyacente, determina sus componentes de la mezcla en cada trama. Su rango ocupa [`0`(`másamplio`)..`1`(`másestrecho`).
`maxhmmpf`	20	(Sólo se utiliza en los modos 4 y 5) Determina el número aproximado de HMMs que pueden permanecer activos en cada trama.
`maxhistpf`	100	(Sólo se utiliza en los modos 4 y 5) Controla el número de los distintos historiales de palabras que son grabados en cada trama.
`maxwpf`	–	Controla el número de palabras distintas que son reconocidas en cualquier trama.
`ci_pbeam`	$1e-80$	Habilita un cómputo en 2 pasadas donde los modelos CI fueron obtenidos primero y luego los modelos CD. Si se utiliza éste haz (beam), únicamente se calcularán los modelos CD que correspondan con los modelos CI dentro del haz (beam), es decir, aquellos que tengan los puntajes máximos en los modelos CI. Su rango ocupa [`0`(`másamplio`)..`1`(`másestrecho`)
`max_cdsenpf`	100000	Es similar a `ci_pbeam` pero el haz (beam) se decide por un número absoluto de senones computados.
`ds`	1	Radio de submuestreo en el cómputo de la trama.
`lw`	9.5	Peso del lenguaje.
`wip`	0.7	Penalización por inserción de palabra.
`Nlextree`	3	(Modo 4 solamente) Número de instancias del árbol léxico. Las entradas a ellas deben estar escalonadas en tiempo.
`rawext`	–	Directorio donde se almacenan los archivos de características.
`feat`	*1s_c_d_dd*	Configuración de las vectores de características. Véase el apéndice C.
`logbase`	1.0003	Base en la cuál son calculadas todas las probabilidades logarítmicas.

Tabla F.3: Parámetros de configuración de `sphinx3_livepretend` (cont.).

Bandera	Default	Descripción
`agc`	max	El tipo de control automático de la ganancia para c_0 que se realizará (max o none).
`beam`	$1.0e-64$	Haz de recorte principal que se aplica a los trifonemas en la búsqueda hacia adelante.
`cb2mllr`	`.1cls.`	Archivo de la relación de la matriz de transformación de senón a MLLR.
`cepdir`	–	Directorio para los archivos cepstral que se usan como entradas. Se fija de antemano en la espcificaciones dadas en el archivo de control.
`cepext`	mfc	Extensión de los archivos cepstral que se usan como entradas. Se fija de antemano en la espcificaciones dadas en el archivo de control.
`cmn`	current	Normalización de la media de los cepstral basado en la pronunciación actual o previa (current o prior).
`ctl`	–	Archivo de control de las pronunciaciones a ser procesadas.
`ctloffset`	0	Número de pronunciaciones que serán saltadas desde el inicio del archivo de control.
`ctlcount`	1000000000	Número de pronunciaciones que serán procesadas después de haber saltado `ctloffset` entradas.
`ctl_mllr`	–	Archivo de control que lista las correpondientes matriz MLLR para una pronunciación.
`dict`	–	Archivo que contiene el diccionario de palabras.
`fdict`	–	Diccionario de palabras de relleno .
`feat`	$1s\_c\_d\_dd$	Configuración de las vectores de características. Véase el apéndice C.
`hmm`	–	Archivos del modelo acústico. Se podría especificar el modelo acústico solamente con la opción `hmm`. Los valores por omisión para los componentes HMM son `means`, `variances`, `mixture_weights`,`transition_matrices`y `mdef`(la definición del modelo).
`hyp`	–	Archivo con los resultados del reconocimiento que sólo contiene palabras.
`hypseg`	–	Archivo de hipótesis resultantes con puntajes y segmentaciones detalladas.
`hypsegfmt`	0	Especifica el formato de segmentación. 0:*Sphinx 3* , 1: *Sphinx 2* o 2:NIST CTM
`insent`	–	Archivo de transcripciones de entradas correspondientes al archivo de control.
`insert`$_s$`il`	1	Indica si se insertan silencios y rellenos opcionales entre las palabras.
`lambda`	–	Archivos de entrada de parámetros de pesos de interpolación de los senones CD/CI.
`log3table`	1	Determina si se utiliza una tabla `logs3` o se calculan los valores en tiempo de ejecución.
`logbase`	1.0003	Base en la cuál son calculadas todas las probabilidades logarítmicas.
`logfn`	–	Archivo de registro. Salida estándar de información y de errores .
`lts_mismatch`	0	Utiliza las reglas letra a sonido (letter-to-sound rules o LTS) del diccionario CMU para generar pronunciaciones para las palabras del modelo del lenguaje que no aparecen en el diccionario. Se asume que el conjunto de fonemas en el archivo de definición del modelo y en el diccionario de pronunciación usan las mismas reglas LTS.
`mdef`	–	Archivo de entrada de definición del modelo .
`mean`	–	Archivo de entrada de las medias de las mezclas Gaussianas.

Tabla F.4: Parámetros de la aplicación `sphinx3_align`.

Bandera	Default	Descripción
`mixw`	–	Archivo de entrada de los pesos mixtos de los senones.
`mixwfloor`	0.0000001	Valor mínimo de los pesos mixtos de los senones. Se aplica al archivo de entrada de los pesos mixtos de los senones
`mllr`	–	Matriz de transformación MLLR que será aplicada a las medias de las mezclas Gaussianas.
`outsent`	–	Archivo de transcripción de salida con las pronunciaciones exactas.
`senmgau`	.cont.	Relación del archivo de mezclas Gaussianas con los senones. Valores semi. o .cont.
`stsegdir`	–	Directorio de salida para los archivos de segmentación (opcional).
`tmat`	–	Archivo de entrada de las matrices de transición.
`tmatfloor`	0.0001	Valor mínimo de las matrices de transición.
`var`	–	Archivo de entrada de las variancias de las mezclas Gaussianas.
`varfloor`	0.0001	Valor mínimo de las variancias de las mezclas Gaussianas.
`varnorm`	no	Indica si se realiza la normalización de la variancia (yes/no)
`wdsegdir`	–	Directorio de salida para los archivos de segmentación de palabras (Opcional).

Tabla F.5: Parámetros de la aplicación `sphinx3_align` (cont.).

Bibliografía

[1] **Anónimo.** ***CMU Sphinx — Get CMU Sphinx at SourceForge.net*** **SourceForge, Inc. EUA. 2009. http://sourceforge.net/projects/cmusphinx/**

[2] **Anónimo.** ***Speech at CMU Web Page*** **CMU. EUA. 2007. http://www.speech.cs.cmu.edu/**

[3] **Becchetti, C. y L. P. Ricotti,** ***Speech Recognition: Theory and C++ Implementation*****, Editorial Wiley, 1999.**

[4] **Benesty, Jacob et ali.** ***Springer Handbook of Speech Processing.*** **Springer. Alemania. 2008.**

[5] **Bernal Bermudez, Jesús et ali,** ***Reconocimiento de voz y Fonética Acústica*****, Alfaomega. México 2000.**

[6] **Chan, Arthur et ali.** ***The Hieroglyphs: Building Speech Applications Using CMU Sphinx and Related Resources.*** **CMU. EUA. 2007. http://www-2.cs.cmu.edu/ archan/documentation/sphinxDocDraft3.pdf**

[7] **Chou, Wu y Biing Hwang Juang.** ***Pattern Recognition in Speech and Language Processing.*** **CRC Press LLC. EUA. 2003.**

[8] **Clarkson, Phillip y Ronald Rosenfeld.** ***Statistical Language Modeling Using the CMU-Cambridge Toolkit.*** **CMU-Cambridge. 1997.**

[9] **Deller, Proakis y Hansen.** ***Discrete-Time Processing of Speech Signals*****, Tercera Edición. Editorial Prentice Hall. New Jersey, U.S.A., 1993**

[10] **Duda, Richard O. Peter E. Hart.** ***Pattern Classification and Scene Analysis*****, Jhon Wiley & Sons, EUA. 1973.**

[11] **Furui, Sadaoki y M. Mohan Sondhi.** ***Advances in Speech Signal Processing.*** **Marcel Dekker, Inc. New York, 1992.**

[12] **Gouvêa, Evandro.,** ***Robust group's Open Source Tutorial. Learning to use the CMU SPHINX Automatic Speech Recognition system.*** **The Sphinx Group, Pittsburg. CMU. 2001.**

[13] **Gouvêa, Evandro et ali.,** ***Sphinx 3 Internals Documentation*** **The Sphinx Group, Pittsburg. CMU. 2009.**

[14] **Herrera Camacho, José Abel.** ***Apuntes del curso de Procesamiento Digital de Voz.*** **Facultad de Ingeniería. UNAM. 1999.**

[15] **Huang, Xuedong, Alex Acero y Hsiao-Wuen Hon,** ***Spoken Language Processing: A Guide to Theory, Algorithm and System Development*****, 1st Prentice Hall, Abril 2001.**

[16] **Huggins-Daines, David,** ***The CMU Sphinx Group Open Source Speech Recognition Engines*** **The Sphinx Group, Pittsburg. CMU. 2006. http://cmusphinx.sourceforge.net/html/cmusphinx.php**

[17] **Hunt, Andrew** ***comp.speech Q6.1: What is speech recognition?*****, Frequently Asked Questions WWW site. Speech Applications Group, Sun Microsystems Laboratories. EUA 1996.**
http://www.speech.cs.cmu.edu/comp.speech/Section6/Q6.1.html

[18] **Jelinek, F.** ***Statistical Methods for Speech Recognition*****, MIT Press, Massachusetts London, England, 1997.**

[19] **Junqua, Jean-Claude y Jean-Paul Haton.** ***Robustness in Automatic Speech Recognition. Fundamentals and Applications.*** **Kluwer Academic Publishers. Massachusets. 1996**

[20] **Jurafsky, Daniel y James H. Martin.** ***Speech and Language Processing. An Introduction to Language Processing, Computational Linguistics and Speech Recognition.*** **Prentice Hall. EUA. 2000.**

[21] **Lee, Kai Fu, Hsiao-Wuen Hon, y Raj Reddy.** ***An Overview of the SPHINX Speech Recognition System*****, IEEE TRANSACTIONS ON ACOUSTICS SPEECH. AND SIGNAL PROCESSING, VOL. 38. NO. I . Enero 1990**

[22] **Lee, Kai. Fu.** ***Automatic Speech Recognition. The Development of Sphinx System.*** **Massachussets. Kluwer Academic Publishers.1989.**

[23] **Lee, Kai Fu, et ali.** ***The SPHINX Speech Recognition System*****, IEEE Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on 1990.**

[24] **Lee, Kai. Fu, et. ali.** ***The Sphinx Speech Recognition System.*** **Computer Science Department. Carnegie Mellon University. Pittsburgh, 1999.**

[25] **Lee, Kai. Fu y Hsiao-Wen Hon.** ***Large-vocabulary speaker-independent continuous speech recognition using HMM*** **International Conference on Acoustics, Speech, and Signal Processing 1988. ICASSP-88, 1988.**

[26] **Manning, C. D. y Schütze, H.** ***Foundations of Statistical Natural Language Processing.*** **6a. reimp. Massachussets. MIT. 2003.**

[27] **Martens, Jean-Pierre ,** ***Continuous Speech Recognition over the Telephone*****, Final Report of COST Action 249, Electronic & Information Systems, Ghent University, May 2000.**

[28] **Moreno, Iván.** ***Manual de etiquetación del Nivel POS (Part-of-Speech) del Corpus DIMEx100.*** **Instituto de Ingeniería. UNAM.2006. http://leibniz.iimas.unam.mx/ luis/DIME/publicaciones/manuales/ Manual-etiquetacion-POS.pdf**

[29] **Nieto Crisóstomo, Omar,** ***Apuntes inéditos de Maestría.*** **Facultad de Ingeniería UNAM. México. 2002.**

[30] **Nieto Crisóstomo, Omar,** ***Diseño de un reconocedor de comandos de voz para el DSP TMS320C6711*****, Tesis de Maestría en Ingeniería Eléctrica, Facultad de Ingeniería, UNAM. 2006.**

[31] **Oppenheim, A. V. and R. W. Schafer,** ***Discrete-Time Signal Processing*****, Prentice Hall, Englewood Cliffs, NJ, 1989.**

[32] **O' Shaughnessy, D.** ***Speech Communications.*** **2a ed. New York. IEEE Press. 2000.**

[33] **Perez Pavón, E. P.** ***Construcción de un Reconocedor de Voz usando Sphinx y el corpus DIMEx100.*** **México. Facultad de Ingeniería. UNAM. 2006.**

[34] **Quilis, Antonio,** ***Tratado de fonología y fonética españolas*****, Segunda edición, Editorial Gredos, Madrid España, 1999.**

[35] **Rabiner, L.R.,** ***A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*****, Proceedings. of IEEE, 1989, Vol. 77 No.2, pp. 257-286.**

[36] **Rabiner, L. R. y R. W. Schafer,** ***Digital Processing of Speech Signals*****,Editorial Prentice-Hall, 1978.**

[37] **Rabiner, Lawrence R. y B. H. Juang, *Fundamentals of Speech Recognition*, Editorial Prentice-Hall, 1993.**

[38] **Ravishankar, Mosur K. *Sphinx-3 s3.X Decoder (X=6)*. Sphinx Speech Group. CMU. EUA. 2004**

[39] **Reyes Cortés, Jaime Alfonso *Aplicación de reconocimiento de voz a comandos de Windows y Word*, Tesis de Ingeniería en Computación, FI, UNAM, 2004.**

[40] **Rudnicky, Alex. *Sphinx Knowledge Base Tool*. CMU. EUA. 1997. http://www.speech.cs.cmu.edu/tools/lmtool-adv.html**

[41] **Seltzer, Michael. *SPHINX III Signal Processing Front End Specification*. Sphinx Speech Group. CMU. EUA. 1999. http://www.cs.cmu.edu/ mseltzer/sphinxman/s3_fe_spec.pdf**

[42] **Singh, Rita. *Manual for the Sphinx-III recognition system*. Sphinx Speech Group. CMU. EUA. 2000. http://www.speech.cs.cmu.edu/sphinxman/**

[43] **Singh, Rita. *Troubleshooting: tools and logfiles*. Sphinx Speech Group. CMU. EUA. 2000. http://www.speech.cs.cmu.edu/sphinxman/fr6.html**

[44] **Singh, Rita. *Decoding*. Sphinx Speech Group. CMU. EUA. 2000. http://www.speech.cs.cmu.edu/sphinxman/**

[45] **Singh, Rita, *Robust group's Open Source Tutorial. Instruction Set for Training* . CMU. Pittsburg. 2001. http://www.speech.cs.cmu.edu/sphinxman/fr4.html**

[46] **Singh, Rita. *Automatic Speech Recognition*, MIT Open Course Ware. MIT. EUA. 2003. http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-345Automatic-Speech-RecognitionSpring2003/ .**

[47] **Singh, Rita. *Sphinx-3 FAQ*. Sphinx Speech Group. CMU. EUA. 2006. http://www.speech.cs.cmu.u/sphinxman/FAQ.html http://www.speech.cs.cmu.edu/sphinxman/FAQ.html**

[48] **Huggins-Daines, David et ali., *The CMU Sphinx Group Open Source Speech Recognition Engines*, The Sphinx Group, Carnegie Mellon, 2001. http://sourceforge.net/projects/cmusphinx/**

[49] **Young, Steve. *Large Vocabulary Continuous Speech Recognition: a Review*. Cambridge University Engineering Department. Cambridge University. 1996.**