

Instituto tecnológico de Massachussets

Departamento de ingeniería eléctrica e informática

6.345 Reconocimiento automático del habla
Primavera 2003

Publicado: 04/04/2003

Entrega :16/04/2003

Trabajo 8

Reconocimiento de voz con sistemas basados en HMM: CMU SPHINX-3 y SPHINX-4

1. Introducción

En esta práctica, usted aprenderá a utilizar un sistema completo de reconocimiento de voz basado en modelos HMM. Este tipo de sistemas, como tantos otros, es un clasificador de patrones estadísticos. Asocia un modelo HMM con cada unidad de sonido. Aprende primero los parámetros de estos modelos HMM y luego utiliza los HMM para hallar la secuencia más probable de unidades de sonido de una determinada señal de voz. El proceso de aprendizaje de los parámetros se denomina *entrenamiento*. El proceso de utilización de éstos, para deducir la secuencia más probable de unidades de una determinada señal, se conoce como *decodificación*, o simplemente reconocimiento. En consecuencia, un sistema de reconocimiento presenta dos componentes principales: un *entrenador* y un *decodificador*. En esta práctica, aprenderá a utilizar el entrenador del sistema SPHINX-3, diseñado en Carnegie Mellon University y escrito en el lenguaje de programación C, además del decodificador del sistema SPHINX-4, un sistema de código abierto de última generación, escrito en el lenguaje de programación JAVA. Dado que SPHINX-4 es un sistema de desarrollo (al que usted también puede contribuir) aprenderá a utilizar el decodificador *en su estado más reciente en el momento de su ejercicio de prácticas*.

El propósito de esta práctica es ayudarle a que se centre en varias cuestiones importantes relacionadas con el uso de un sistema de ASR basado en HMM y a que obtenga una visión de estado actual de la tecnología de reconocimiento basada en HMM.

2. Lo que le facilitaremos

Le facilitaremos todo lo que necesite de fuentes externas para el entrenamiento y la decodificación. No obstante, tendrá que entrenar sus propios modelos acústicos para decodificar. Este ejercicio de prácticas le guiará en los pasos necesarios.

Componentes facilitados para el entrenamiento

El entrenador SPHINX-3 que le facilitaremos consta de un grupo de programas en C que se han compilado para su sistema operativo (Linux). Para el entrenamiento sólo debe utilizar los ejecutables precompilados del sistema. Le damos el código fuente para su información, para quienes tengan curiosidad por los aspectos del software de SPHINX o deseen implementar cualquier pequeña modificación en el código basándose en sus ideas. En esta práctica no es necesario que trabaje con el código fuente.

El entrenador aprende los parámetros de los modelos de las unidades de sonido utilizando un grupo de señales de muestras de habla. Estas señales constan de una **base de datos de entrenamiento**. Le proporcionaremos una base de datos de entrenamiento compuesta por 1600 señales de voz. El entrenador necesita también que le informen de qué unidades de sonido deben aprender los parámetros, y por lo menos, una secuencia en las que éstas podrían haber aparecido en cada señal de voz de su base de datos de entrenamiento. Esta información se le facilita al entrenador a través de un archivo denominado **archivo de transcripción**, en el que la secuencia de palabras y sonidos no discursivos se escriben exactamente como aparecían en la señal de voz, seguidos por una etiqueta que puede utilizarse para asociar esta secuencia con la señal de voz correspondiente.

Entonces, el entrenador busca un **diccionario** que asocie cada palabra con al menos una secuencia de unidades de sonido, para derivar una secuencia de unidades de sonido asociadas con cada señal. Por tanto, además de las señales de voz, también le daremos un grupo de transcripciones para la base de datos (en un único archivo) y dos diccionarios, uno en el que las palabras legítimas de la lengua sean secuencias asociadas de unidades de sonido (o unidades de subpalabra), y otro en el que los sonidos no discursivos estén asociados con las correspondientes unidades de sonido discursivas o no discursivas. Nos referiremos al primero como el **diccionario del lenguaje** y al segundo como el **diccionario de relleno**.

Los componentes proporcionados para el entrenamiento son:

- 1- Los ejecutables del entrenador.
- 2- Señales acústicas para el entrenamiento (datos de entrenamiento)
- 3- El archivo de transcripción correspondiente.
- 4- El diccionario del lenguaje
- 5- El diccionario de relleno.

Componentes facilitados para la decodificación

El decodificador SPHINX-4 que le facilitaremos consta de un grupo de programas en JAVA que han sido precompilados. Los ejecutables de JAVA son independientes del tipo de sistema operativo utilizado por su máquina. Si quiere ejecutar sus decodificaciones en una máquina distinta con un sistema operativo distinto, simplemente copie los archivos SPHINX-4 en esa máquina, asegúrese de que tiene instalado el compilador de JAVA y ejecute las decodificaciones.

Aunque un decodificador consta de un grupo de programas como el entrenador, la decodificación se realiza normalmente mediante un ejecutable simple, al que usted le pasa un grupo de entradas. Las entradas necesarias son: los modelos acústicos entrenados, un archivo índice modelo, un modelo de lenguaje, un diccionario del lenguaje, un diccionario de relleno y un conjunto de señales acústicas que deben ser reconocidas. Los datos que van a ser reconocidos se conocen normalmente como **datos de prueba**. Los componentes para la decodificación son:

- 1- El diccionario del lenguaje.
- 2- El diccionario de relleno.
- 3- El modelo de lenguaje.
- 4- Los datos de prueba.

Además de estos componentes, será necesario que dé los **modelos acústicos** que haya entrenado como entradas al decodificador. Mientras entrena los modelos acústicos, el entrenador generará archivos **índice-modelo** adecuadamente nombrados. Un archivo índice modelo contiene simplemente identificadores numéricos para cada estado de cada HMM, utilizados por el entrenador y el decodificador para acceder a los grupos correctos de parámetros, para aquellos estados HMM a partir de los archivos que contienen sus modelos acústicos entrenados.

Con cualquier grupo determinado de modelos acústicos, el archivo índice modelo correspondiente debe utilizarse para la decodificación. Si quisiera saber más sobre la estructura del archivo índice modelo, encontrará una descripción en la siguiente URL:

<http://www.cs.cmu.edu/~rsingh/sphinxman/fr4.html> bajo el enlace *Creating the CI model definition file* (Crear el archivo de definición del modelo CI).

3. Cómo montar su sistema y cómo familiarizarse con él durante una ejecución preliminar

Para montar su sistema, acceda a la máquina con su identificador de usuario. Encontrará la contraseña en la nota que se adjunta a este documento. Una vez dentro, escriba el siguiente comando desde la línea de comandos:

```
start_lab8.cmd
```

Esto creará un nuevo directorio llamado **lab8/** e instalará en él todos los archivos necesarios para que complete este ejercicio de prácticas. Dentro del directorio **lab8/**, encontrará dos directorios llamados **SPHINX3/** y **sphinx4**. El directorio **SPHINX3** contiene los siguientes subdirectorios:

- **s3trainer/**: El código fuente del entrenador SPHINX-3 y los ejecutables.
- **lists/**: El grupo de todos los archivos de entrenamiento facilitados por el usuario. Usted entrenará modelos acústicos utilizando una base de datos llamada base de datos de gestión de recursos (RM). Si está interesado, puede encontrar más

información sobre esta base de datos en <http://www ldc.upenn.edu/Catalog/LDC93S3B.html>. En el directorio **lists/** encontrará una lista de enunciados de entrenamiento (**train.wavlist**) y un grupo de enunciados de prueba (**test.wavlist**) de esta base de datos.

Si desea escuchar algunos de los archivos enumerados en ***.wavlist**, utilice el comando **waveform_play** [**nombre del archivo de la forma de onda**]. Además de ***.wavlist**, también encontrará un archivo de transcripción (**RM.1600.trans**) con transcripciones correspondientes a los archivos listados en **train.wavlist**, un diccionario del lenguaje para el entrenamiento (**RM.dictionary**), un diccionario de relleno (**filler.dict**) y una lista de fonos (**RM.phonelist**) que contiene todas las unidades de subpalabra utilizadas en el diccionario del lenguaje, así como un símbolo adicional **SIL** para el silencio.

- **c_scripts/** : El grupo de scripts que utilizará para entrenar modelos HMM de densidad continua para la base de datos RM. Este directorio también contiene un script que usted utilizará para computar rasgos (MFCC) para su entrenamiento y probar enunciados. Recuerde que el sistema de reconocimiento se entrenará con estos rasgos y no directamente con las formas de onda.

El directorio Sphinx4 contiene muchos subdirectorios, de los cuales, los específicos que se muestran a continuación, le serán muy útiles para esta práctica:

- **edu/** : Este contiene el código fuente de Java para el decodificador SPHINX-4.
- **classes/** : Este contiene las clases de Java compiladas desde el código del directorio.
- **edu/**. Estos serán utilizados por la máquina virtual de Java (JVM) instalada en su ordenador para ejecutar su decodificador.
- **tests/performance/resource-management/** : Este es el directorio en el que se llevará a cabo la decodificación. Contiene un archivo (**rm1.props**) que indicará al decodificador los modelos acústicos que usted entrene, un modelo que posteriormente ejecutará el decodificador SPHINX-4 (**Makefile**) a través de un comando *make* sencillo.
- **tests/performance/resource-management/RM_models_and_otherfiles/**: Este contiene una lista de 100 archivos de voz que usted decodificará (**s4.100.ctl**), un modelo de lenguaje (**RM.2880.bigram.arpa**), un diccionario (**400.dict**) y un diccionario de relleno (**filler.dict**), todo ello para la decodificación. En esta práctica, no es necesario que modifique el archivo del modelo de lenguaje, pero *puede* modificar el diccionario utilizado para la decodificación. Tenga en cuenta que el diccionario destinado a la decodificación es distinto al utilizado para el entrenamiento. Esto es normalmente así en los sistemas de reconocimiento de voz. El diccionario de decodificación está elaborado con mucho más esmero que el de entrenamiento, para minimizar la posibilidad de confusión o para acomodar nuevas palabras, pronunciaciones y otras condiciones.
- **tests/live/** : Este contiene archivos que le harán falta para realizar la *decodificación en vivo*. En este modo de decodificación, puede hablar por los micrófonos de su máquina, y su voz será directamente leída y reconocida por el decodificador del dispositivo de audio de la misma. Las hipótesis del reconocimiento se mostrarán en

la pantalla a través de una interfaz gráfica GUI de Java. Los archivos de este directorio se organizarán para que pueda llevar a cabo la decodificación en vivo. Para su información, el archivo **rm1_live.props** apunta a los modelos acústicos, diccionarios de decodificación, *etc.*, y el archivo **Makefile** ejecuta la decodificación en vivo mediante el comando simple *make live*.

Cómo realizar una vuelta de entrenamiento preliminar

La primera cosa que debe hacer es computar los archivos de rasgos de los datos de voz que le hemos facilitados para el entrenamiento. Para llevar esto a cabo, entre en el directorio **SPHINX3/c_scripts/** y escriba el siguiente comando en la línea de comandos:

```
compute_mfcc_train.csh ../lists/train.wavlist
```

Este script computará para cada enunciado, una secuencia de coeficientes cepstrales de frecuencia Mel (MFCC) de 13 dimensiones. Este paso lleva aproximadamente 10 minutos en una red rápida, pero puede llevarle hasta 40 minutos si la red es lenta. Los MFCC se situarán automáticamente en un directorio llamado **SPHINX3/feature_files/**. Observe que el tipo de vectores característicos que utilice para el entrenamiento y el reconocimiento, fuera de esta práctica, no está limitado a coeficientes MFCC. Podría utilizar cualquier tipo de rasgos, incluso aquellos derivados de fenómenos simultáneos al habla, como las grabaciones de video. SPHINX-3 y SPHINX-4 pueden utilizar rasgos de cualquier tipo o dimensionalidad. Los coeficientes MFCC son conocidos actualmente como la mejor parametrización de rasgos simples para el funcionamiento de un óptimo reconocimiento de voz en sistemas basados en modelos HMM, bajo las mayores condiciones acústicas. Una vez que vuelve a aparecer la ventana de símbolo del sistema, ya puede empezar a entrenar el sistema. No obstante, antes de hacer esto, compute también los rasgos para los datos de prueba. Esto no es realmente necesario ahora, pero es bastante sencillo de hacer si escribe el siguiente comando desde la línea de comandos:

```
compute_mfcc_test.csh ../lists/test.wavlist
```

Esto colocará los archivos MFCC que usted utilizará para la decodificación, en un lugar específico al que el decodificador podrá acceder automáticamente.

Para entrenar el sistema entre en el directorio **SPHINX3/c_scripts**. Dentro de este directorio, hay 5 directorios enumerados secuencialmente desde 01 hasta 05. Entre en cada directorio comenzando por **01** y ejecute el script llamado **slave*.csh** dentro de ese mismo directorio. Estos scripts lanzarán trabajos en el sistema en serie SLS, y cada trabajo llevará unos cuantos minutos para su ejecución.

Antes de que ejecute cualquier script, fíjese en los contenidos del directorio de SPHINX3. Después de ejecutar cada **slave*.csh**, fíjese en los contenidos de nuevo.

Se habrán creado varios directorios nuevos. Estos directorios contienen archivos que irán generándose en el curso de su entrenamiento. En este momento no es necesario que conozca el contenido de estos directorios, aunque algunos de sus nombres pueden ser autoexplicativos y puede explorarlos si siente curiosidad. Después de lanzar el primer **slave*.csh**, observe la salida de la pantalla hasta que vuelva a aparecer la ventana de símbolo del sistema. Sólo entonces, entre en el nuevo directorio (02) de la secuencia especificada y lance **slave*.csh** en ese directorio. Repita este proceso hasta que haya ejecutado **slave*.csh** en los cinco directorios.

En este momento podría haber completado el entrenamiento. Encontrará los modelos acústicos en un directorio llamado **SPHINX3/model_parameters/RM.cd_continuous_8gau**. En este directorio hay cuatro archivos que constituyen sus modelos acústicos: *means*, *variances*, *transition-matrixes* y *mixture-weights* (medias, varianzas, matrices de transición y pesos de mezcla). Estos son archivos binarios. Sus versiones ASCII *.**ascii**, también se encuentran en el mismo directorio y serán utilizadas por el decodificador SPHINX-4. Además de éstas, encontrará un archivo para estos modelos llamado **RM.1000.mdef** en el directorio **SPHINX3/model_architecture/**. El sistema utiliza este archivo para asociar el grupo de parámetros HMM adecuados, con el modelo HMM para cada unidad de sonido que usted esté modelando.

Más adelante en este documento, explicaremos más detenidamente algunos aspectos modo en vivo del proceso de entrenamiento.

Cómo realizar una decodificación preliminar

Esto es sencillo. Vaya al directorio **sphinx4/tests/performance/resource_management/**y desde la línea de comandos, escriba:

make batchmode_bigram

Esto mostrará en pantalla la salida del reconocedor. Puede también guardarla al mismo tiempo en algún archivo **filename.log** con el siguiente comando:

make batchmode_bigram |& tee filename.log

Como se mencionó antes, SPHINX-4 es un sistema de código abierto que aún está bajo desarrollo. Está mejorando activamente en muchos aspectos a medida que el tiempo avanza, y puede descargarse en cualquier momento la última versión del sistema para su uso personal. Para hacerlo, visite la página **http://cmusphinx.sourceforge.net** y siga las instrucciones de descarga escritas allí. Puede descargar también un compilador de Java desde **http://java.sun.com/j2se/1.4/**.

Para su ejercicio de prácticas, el decodificador SPHINX-4 está programado para funcionar en *modo en serie*, donde opera sobre archivos de rasgos precomputados guardados en el disco. El decodificador puede ejecutarse también en otros modos. En *modo en vivo simulado*, un decodificador utiliza señales de voz guardadas en un disco y las decodifica en bloques pequeños, computando rasgos de los bloques, utilizándolos y luego descartándolos. En la decodificación en *modo en vivo*, la voz capturada de forma activa por el dispositivo de audio de su máquina, es leída directamente por el decodificador y reconocida.

Una interfaz GUI sencilla que demuestra todo esto, está disponible en:

http://cmusphinx.sourceforge.net/cgi-

bin/twiki/view/Sphinx4/HowToBuildAndRunSphinx4#_Live_Mode_Demo_

El archivo de registro que usted acaba de generar, **filename.log**, contiene las hipótesis decodificadas y más información útil, como precisión en el reconocimiento y tasas de error. El formato de este archivo se explica más detenidamente en este documento, pero por ahora, fíjese en las siguientes líneas

Palabras: 832 Correspondencias : 786 WER: 6.971%
Oraciones: 100 Correspondencias: 68Oraciónexact.: 68.000%

Estas líneas (y muchas otras) aparecen después de cada archivo que es decodificado. Los valores contra cada etiqueta aumentan a medida que se decodifican más archivos (u oraciones). La etiqueta **oraciones: 100** implica que hasta ahora se han decodificado 100 oraciones. La tasa de error por palabra esta vez parece ser

WER: 6.971%. Fíjese en este número. En su archivo de registro, la tasa WER de la oración nº 100 debería estar dentro de ± 1 de este número. Si no es así, informe de ello. Puede que se haya producido algún problema en su entrenamiento preliminar.

4. Herramientas variadas

Le facilitamos dos herramientas útiles junto con los ejecutables del entrenamiento en el directorio **SPHINX3/s3trainer/bin.linux/**.

Las herramientas se describen más abajo. Las instrucciones sobre cómo utilizar estas herramientas se encuentran en los archivos **toolname_instructions** que usted encontrará junto con los ejecutables.

1. Herramienta de análisis de frecuencia de fonos y trifonos: Este es el **mk_mdef_gen** ejecutable. Puede utilizarlo para computar las frecuencias relativas de aparición de sus unidades básicas de sonido (fonos y trifonos) en la base de datos de entrenamiento. Dado que los HMM son modelos estadísticos, su objetivo es diseñar unidades básicas de sonido que aparezcan frecuentemente para que sus modelos se calculen adecuadamente, mientras que se mantiene suficiente información para minimizar confusiones entre palabras.
Esta cuestión queda explicada más detalladamente en el Apéndice 1.
2. Herramienta para ver los archivos MFCC: Este es el **cepview** ejecutable. Puede observar los cepstros en cualquier archivo de rasgos, siguiendo las instrucciones de este archivo de instrucciones de herramientas.

5. Qué esperamos en esta práctica

El objetivo principal de esta práctica es que adquiera conciencia razonable del modo de funcionamiento de los sistemas típicos basados en HMM, y de la respuesta de éstos en diversos entornos. Debe seleccionar un modo de “seguimiento” del funcionamiento del sistema, con respecto a algún aspecto del mismo que afecte al rendimiento en el reconocimiento. A continuación se describen algunos aspectos importantes que puede examinar. Elija uno de ellos, como mucho dos, y experimente con ellos, piense y estúdielos del modo que le parezca más útil. Luego simplemente anote en una o dos páginas, cómo examinó el sistema y qué averiguó. Por ejemplo, podría crear un grafo o una tabla, y podría tratar de explicar las tendencias observadas. No olvide que el reconocimiento de voz es un problema de ingeniería complejo y, en esta práctica, no es necesario que relacione totalmente los aspectos que ha estudiado con una gran parte del resto del sistema. Simplemente trate de hacerlo lo mejor posible.

6. Cómo entrenar y cuestiones de entrenamiento claves

En este momento ya se encuentra preparado para comenzar su ejercicio de prácticas. Será necesario nombrar primero cada vuelta de entrenamiento y decodificación. Nos referiremos al nombre del experimento que usted elija con **[experimentname]**. Por ejemplo, puede nombrar al primer experimento *exp1* y al segundo *exp2*, etc. Su elección del nombre del experimento **[experimentname]** se añadirá automáticamente a todos los archivos pertinentes creados bajo ese experimento, para una sencilla identificación. Todos los directorios y archivos requeridos para este experimento se encontrarán en un directorio llamado **lab8/[experimentname]**. Para comenzar un nuevo experimento, entre en el directorio **lab8** y utilice el comando

SPHINX3/c_scripts/create_newexpt.csh [experimentname]

Esto creará un sistema parecido al del entrenamiento preliminar y al de pruebas, en un directorio llamado **experimentname/** en su directorio base. Después de esto, debe trabajar de pleno dentro de este directorio **[experimentname]**.

Su ejercicio de prácticas comienza con el entrenamiento del sistema utilizando archivos de rasgos MFCC que ya ha computado durante la vuelta preliminar. Sin embargo, cuando entrene esta vez, puede tomar ciertas decisiones basándose en lo que ha aprendido hasta ahora en el trabajo realizado para el curso y en la información que le hemos facilitado en este documento. Las decisiones que tome afectarán a la calidad de los modelos que entrene, y por tanto, al rendimiento del reconocimiento del sistema.

Lo siguiente son algunos aspectos importantes que puede observar con relación al entrenamiento:

1. Decida qué unidades de sonido va a permitir que el sistema entrene. Puede modificar la calidad de sus modelos acústicos eligiendo distintas unidades de sonido. Para hacer esto, observe el diccionario de entrenamiento **[experimentname]/lists/RM.dictionary** y el diccionario de relleno **[experimentname]/lists/filler.dict**, y fíjese en las unidades de sonido de ambos. Una lista de los sonidos que aparecen en estos diccionarios también se encuentra escrita en el archivo **[experimentname]/lists/RM.phonelist**. Existen muchos modos de enfocar la cuestión relativa a las unidades de sonido óptimas:
 - Estudie los diccionarios para ver si las unidades de sonido se utilizan coherentemente y se prestan a una mínima confusión. Observe el **apéndice 1** para una explicación. Si es que no, cambie el uso a apropiado.
 - Dados algunos grupos de unidades de sonidos, puede comprobar si estas unidades, y los trifonos que pueden formar (para los que construirá modelos en última instancia), están bien representados en los datos de entrenamiento. Es importante que las unidades de sonido que se estén modelando, estén adecuadamente representadas en los datos de entrenamiento para estimar de forma fiable los parámetros estadísticos de sus HMM. Para estudiar su frecuencia de aparición en los datos, puede utilizar la herramienta **mk_mdef_gen**. Las instrucciones para el uso de esta herramienta están disponibles en su correspondiente archivo de instrucción.

Puede modificar la estadística de aparición de trifonos fácilmente, cambiando, fusionando o dividiendo unidades de sonidos existentes en los diccionarios. Mediante la fusión de unidades de sonido, nos referimos al agrupamiento de dos unidades de sonido distintas en una única entidad. Por ejemplo, es posible que desee modelar los sonidos “Z” y “S” como una unidad simple (en vez de mantenerlas como unidades separadas).

Para fusionar estas unidades, que están representadas por los símbolos Z y S en el diccionario de la lengua determinado, sustituya simplemente todas los casos de Z y S en el diccionario, por un símbolo común (que podría ser Z_S o un símbolo completamente nuevo). Mediante la división de unidades de sonido, estamos implicando la introducción de nuevas y múltiples unidades de sonido en lugar de una simple. Este es el proceso inverso de la fusión. Por ejemplo, si encuentra un diccionario del lenguaje donde todas los casos de los sonidos Z y S estuviesen representadas por el mismo símbolo, es posible que quisiese sustituir este símbolo por Z en algunas palabras y por S en otras. Las unidades de sonido también pueden reestructurarse mediante la agrupación de secuencias específicas de sonidos en un único sonido. Por ejemplo, podría modificar todas los casos de la secuencia "IX D" para obtener un único sonido IX_D. Esto introduciría un nuevo símbolo en el diccionario, mientras que mantendría todos los que anteriormente existían. El número de unidades de sonido aumenta eficazmente en uno en este caso. Existen otras técnicas utilizadas para redefinir unidades de sonido para una tarea determinada. Si se le ocurre cualquier otro modo de redefinir diccionarios o unidades de sonido que pueda justificar correctamente, le animamos a que lo intente.

Una vez que ha rediseñado sus unidades, modifique en consecuencia la **RM.phonelist** del archivo. Asegúrese de que no tiene falsos espacios vacíos o líneas en este archivo. *Fíjese de nuevo en que es posible que usted pase por alto totalmente este procedimiento de diseño o utilice la lista de fonos y diccionarios tal y como le han sido facilitados.* Fíjese en que desde que ha modificado las unidades de sonido de su diccionario de entrenamiento, *debe* modificarlas también en el diccionario de decodificación, ya que el sistema puede reconocer sólo las unidades de sonido que ha aprendido. Así que haga cambios parecidos en el diccionario de decodificación

[experimentNAME]/sphinx4/tests/performance/resource_management/RM_models_and_otherfiles/400.dict.

Una vez que haya fijado sus diccionarios y el archivo de la lista de fonos, edite el archivo **variables.def** en **[experimentname]/c_scripts/** para introducir los siguientes parámetros de entrenamiento:

- **set dictionary** = su diccionario de entrenamiento con ruta completa (no cambie si ha decidido no modificar el diccionario).
- **set fillerdict** = su diccionario de relleno con ruta completa (no cambie si ha decidido no modificar el diccionario).
- **set phonefile** = su lista de fonos con ruta completa (no cambie si ha decidido no modificar el diccionario).

2. Otro aspecto que puede estudiar es el efecto de la variación en el número de parámetros que el sistema debe entrenar desde los datos de entrenamiento determinados. Esto se logra fácilmente reubicando los siguientes indicadores en **variables.def set statesperhmm** = Esto se ajusta a 3 ó 5 en sistemas estándar. Trate de ajustarlo para otros valores (se recomienda permanecer en el umbral de 3-9). El número de estados de un HMM está relacionado con las características variables en el tiempo de las unidades de sonido. Las unidades de sonido que presentan una gran variación temporal necesitan más estados que las representen. La naturaleza de la variación temporal de los sonidos está también en parte capturada por la variable "*skipstate*" (saltar estado) que se describe a continuación.

- **set skipstate** = ajuste esta a "no" o "sí" sin las comillas dobles. Esta variable controla la topología de sus HMM. Cuando se ajusta a "sí", permite que los HMM salten estados. Sin embargo, fíjese en que la topología HMM utilizada en este sistema es una topología Bakis estricta de izquierda a derecha. Si ajusta esta variable a "no", cualquier estado determinado puede sólo hacer transición al próximo estado. En todos los casos, se permiten autotransiciones. Observe las figuras del **apéndice 2** para más consulta. Encontrará el archivo de topología HMM en el directorio llamado **model_architecture/** en su actual directorio base (**experimentname**).
- **set gaussiansperstate** = ajuste este a cualquier número desde 4 hasta 8. No es aconsejable que pase de 8 por el pequeño grupo de datos de entrenamiento que le hemos facilitado. La distribución de cada estado de cada HMM se modela por una mezcla de gaussianas. Esta variable determina el número de gaussianas de esta mezcla. El número de parámetros HMM que van a estimarse, aumenta a medida que sube el número de gaussianas de la mezcla. Por tanto, aumentar el valor de esta variable puede provocar que haya menos datos disponibles para estimar los parámetros de cada gaussiana. Sin embargo, aumentar su valor también produce modelos más elegantes, que pueden conducir a un mejor reconocimiento. Por tanto, a estas alturas es necesario pensar con criterio en el valor de

esta variable, teniendo en cuenta estas dos cuestiones. Recuerde que es posible superar el problema de insuficiencia de datos compartiendo las mezclas de gaussianas entre muchos estados HMM. Cuando múltiples estados HMM comparten la misma mezcla de gaussianas, se habla de estados compartidos o atados. Estos estados compartidos se conocen como estados atados (también denominados senones). El número de mezclas que usted entrene será, en última instancia, exactamente igual al número de estados atados que usted especifique, que a su vez puede estar controlado por el parámetro "n_tied_states" descrito anteriormente.

- **set n_tied_states** = Ajuste este número a cualquier valor entre 500 y 2500. Esta variable le permite especificar el número total de distribuciones de estado compartidas de su grupo final de HMM entrenados (sus modelos acústicos). Los estados se comparten para superar problemas de insuficiencia de datos para cualquier estado de cualquier HMM. La distribución se realiza de tal modo que se mantenga la "individualidad" de cada HMM, en el sentido de que sólo los estados con las distribuciones más parecidas son "atados". El parámetro **n_tied_states** controla el grado de atadura. Si éste es pequeño, es posible que un gran número de estados sin parecido alguno pueden estar atados, provocando una reducción en el rendimiento del reconocimiento. Por otro lado, si este parámetro es muy grande, pueden existir datos insuficientes para aprender los parámetros de las mezclas de gaussianas para todos los estados atados. (Puede encontrar una explicación relativa a las ataduras de estados en el **apéndice 3**). Si siente curiosidad, puede observar qué estados ha atado el sistema, echándole un vistazo al archivo `SPHINX3/model_architecture/[experimentname].n_tied_states.mdef` . y comparándolo con el archivo `SPHINX3/model_architecture/[experimentname].untied.mdef`. Estos archivos enumeran los fonos y trifonos para los que usted entrena los modelos, y asigna identificadores numéricos a cada estado de sus HMM. No obstante, cuando usted cambia los **n_tied_states**, antes de decodificar, asegúrese de que el archivo `[experimentname]/sphinx4/tests/performance/resource_management/rm1.props` señala al archivo mdef correcto y a los archivos del modelo acústico para su experimento.

3. Finalmente, puede controlar el grado de ajuste (o sobreajuste) de los modelos estimados, controlando el número de pases que el algoritmo de Baum-Welch realiza con los datos de entrenamiento para entrenar los modelos. Este aspecto se controla ajustando los siguientes valores de indicación en **variables.def**:

- **set convergence_ratio** = ajusta esto a un número desde 0.1 a 0.001. Este número es la proporción entre la diferencia en probabilidad de la iteración actual y la anterior de Baum-Welch, y la probabilidad total de la iteración anterior. Fíjese aquí que la tasa de convergencia depende de varios factores como la inicialización, el número total de parámetros que se estiman, la cantidad total de datos de entrenamiento y la variabilidad inherente en las características de los datos de entrenamiento. Cuántas más iteraciones de Baum-Welch ejecute, mejor aprenderá las distribuciones de sus datos. Sin embargo, los cambios menores que se obtengan en las mayores iteraciones del algoritmo de Baum-Welch, pueden no afectar al funcionamiento del sistema. Teniendo esto en cuenta, decida cuántas iteraciones desea que su entrenamiento de Baum-Welch ejecute en cada etapa. Esta es una decisión subjetiva que debe tomarse basándose en la primera ratio de convergencia que encontrará escrita al final del archivo de registro para la segunda iteración de su entrenamiento de Baum-Welch
(SPHINX3/logdir/0*/[experimentname].*.2.norm.log. Normalmente, son suficientes de 5 a 15 iteraciones, dependiendo de la cantidad de datos que tenga. No entrene más de 15 iteraciones.
- **set maxiter** = ajuste este a un número entero entre 5 y 15. Esto limita el número de iteraciones de Baum-Welch al valor de **maxiter**.

Una vez que ha realizado todos los cambios deseados, debe entrenar un nuevo conjunto de modelos. Puede lograr esto volviendo a ejecutar todos los scripts **slave*.csh** de los directorios [experimentname]/c_scripts/01* a través de [experimentname]/c_scripts/05*.

7. Cómo decodificar y cuestiones relativas a la decodificación de la clave

Para decodificar, simplemente vaya al directorio [experimentname]/sphinx4/tests/performance/resource_management/, asegúrese de que ha hecho cualquier cambio necesario en el diccionario de decodificación en **RM_models_and_otherfiles/400.dict**. Para asociar las modificaciones realizadas en su diccionario de entrenamiento, asegúrese de que los archivos del modelo acústico que apuntan al archivo **rm1.props** son correctos, y luego, desde la línea de comandos, escriba: **make batchmode_bigram |& tee filename.log**

Las cuestiones más importantes sobre el uso eficiente de un decodificador, son aquellas relacionadas con su velocidad de decodificación, uso de memoria y consumo de potencia. Dado que para esta práctica estará trabajando en ordenadores fijos, el consumo de potencia no es una cuestión crucial. El rendimiento de un decodificador depende normalmente de un equilibrio entre entornos creados para minimizar el uso recursos y maximizar la velocidad, aquellos necesarios para acomodar un tamaño y complejidad concretos de los modelos acústicos, modelos de lenguaje y diccionarios, y la precisión en el reconocimiento. Muchos

de estos balances deben decidirse en la etapa de entrenamiento. El resto de modificaciones de los ajustes se realizan en el decodificador. En esta práctica tendremos en cuenta algunos ajustes muy básicos del decodificador que afectan a las necesidades logísticas, velocidad y rendimiento en el reconocimiento. Fíjese en que mientras en muchos casos, el rendimiento es lo que importa, en la actualidad, la mayoría de los sistemas de reconocimiento de voz deben utilizarse en muchos escenarios que prestan también una rigurosa atención a cuestiones de memoria y velocidad.

Estas cuestiones se han puesto muy de moda en los últimos años. Así que antes de utilizar cualquier sistema de reconocimiento de voz, es buena idea hacer un balance de sus recursos disponibles, de manera que pueda utilizarlos prudentemente. SPHINX-4 está escrito con JAVA. Este es un lenguaje orientado a objetos como C++, pero se ejecuta a través de un interpretador archivo-objeto intermedio conocido como la máquina virtual de Java (JVM), que convierte sus archivos de objeto compilados en un formato interno que se ejecutará en su sistema operativo específico. Hasta donde a usted (usuario) le concierne, el código que haya compilado en JAVA en cualquier máquina, se ejecuta con cualquier otro sistema operativo distinto, dado que tienen un JVM instalado. JAVA presenta algo adicional – no utiliza punteros, y no es necesario que libere memoria explícitamente en su código. JAVA posee un *Garbage Collector* (GC) (recolector de basura) que periódicamente siente qué partes de la memoria distribuida no están siendo utilizadas, y las libera. Es lógico pensar que trabajar con un interpretador intermedio, y con la revisión continua del GC, ralentizará cualquier aplicación de JAVA. Sin embargo, en buena parte, esto no se ha convertido en un problema, por varias razones.

Existen algunas cosas que puede hacer para conseguir memoria eficazmente. Puede especificar los *heapsizes* (reservas de tamaño) que usted desea que el decodificador utilice a través de los indicadores **GC_FLAGS** en el **Makefile**. El valor de **-ms** establece la *starting heapsize* (reserva de tamaño inicial) y el valor de **-mx** ajusta la *maximum heapsize* (reserva de tamaño máxima). Una *heapsize* (reserva de tamaño) en JAVA está relacionada con la cantidad de memoria que JVM puede distribuir en sus programas. Afecta a la acción del GC. Podría servir para ayudar a averiguar algo más sobre la CPU y los recursos de memoria de su máquina, para obtener una idea de donde se encuentra en relación con estos ajustes del indicador. No obstante, esto no pretende ser una parte de su trabajo de prácticas. La memoria y la velocidad pueden conseguirse también de muchos otros modos. SPHINX-4 no se ha optimizado aún para memoria y velocidad, y usted puede contribuir al sistema en estas áreas después de la práctica, si está interesado.

Listas activas y anchos de haz: Existe otro modo en el que tanto el uso de la memoria como la velocidad pueden ser controlados (y esto es aplicable a decodificadores escritos en cualquier lenguaje): mediante la especificación adecuada del tamaño de lo que se conocen como *listas activas*.

En cualquier instante durante la búsqueda, una lista activa se compone de aquellas densidades gaussianas que deben ser computadas explícitamente por el decodificador, dado el vector de datos actual. Este es un subconjunto de todas las gaussianas presentes en los modelos acústicos que el decodificador utiliza, y han sido alcanzadas por trayectorias actuales del trellis. Puede controlar el tamaño de la lista activa mediante los indicadores

```
edu.cmu.sphinx.search.ActiveList.absoluteBeamWidth  
edu.cmu.sphinx.search.ActiveList.relativeBeamWidth
```

en el archivo **rm1.props**. El ancho de haz absoluto es convencionalmente un número fijo, y define el tamaño máximo de la lista activa. El ancho de haz relativo varía de un instante en el tiempo a otro, y se define en relación con el nodo de puntuación máxima en el trellis del decodificador en el momento determinado. En sphinx4, es una puntuación umbral, que es un porcentaje de la puntuación máxima actual. A los nodos que presentan una puntuación menor que este umbral, no se les permite propagarse más. Las listas activas se describen en <http://cmusphinx.sourceforge.net/cgi-bin/twiki/view/Sphinx4/HeapActiveList> . Puede encontrar una descripción de anchos de haz en <http://cmusphinx.sourceforge.net/cgi-bin/twiki/view/Sphinx4/Sphinx4Properties>.

Peso del lenguaje: El peso del lenguaje es un parámetro importante en cualquier decodificador. En SPHINX-4, el indicador para esto es

`edu.cmu.sphinx.search.BreadthFirstSearchManager.languageWeight`, en **rm1.props**. Un valor entre 6 y 13 es estándar, y por defecto, el peso del lenguaje no se aplica. El modelo de lenguaje y el peso del lenguaje se describen en el **apéndice 4**. Recuerde que el peso del lenguaje decide qué importancia relativa le dará a las probabilidades acústicas reales de las palabras de la hipótesis. Un peso de lenguaje bajo da más libertad a las palabras con mayores probabilidades acústicas para que sean planteadas como hipótesis, a riesgo de plantear como hipótesis palabras falsas. Puede decodificar varias veces con distintos pesos de lenguaje, sin volver a entrenar los modelos acústicos, para decidir lo que es mejor para usted.

Multa por inserción de palabra: La multa por inserción es un parámetro heurístico importante en cualquier algoritmo de programación dinámica. El decodificador no es la excepción. El indicador para esto es `edu.cmu.sphinx.search.Linguist.wordInsertionProbability` en **rm1.props**. Es el número que decide cuánta multa aplicar a una nueva palabra durante la búsqueda. Si las nuevas palabras no estuviesen penalizadas, el decodificador tendería a plantear como hipótesis las palabras menores posibles, ya que cada nueva palabra insertada produce un aumento adicional en la puntuación de cualquier trayectoria, como resultado de la inclusión de la probabilidad del lenguaje de la palabra insertada del modelo de lenguaje.

Uso de otros modelos acústicos: Finalmente, fíjese en que puede utilizar cualquiera de los modelos intermedios que generó durante el entrenamiento, para la decodificación. Asegúrese de utilizar el archivo índice del modelo adecuado con cada grupo de modelos. El archivo índice modelo es el mismo para los modelos cd, pero es distinto para los modelos ci y cd desatados. Encontrará todos los modelos acústicos en **SPHINX3/model_parameters/**, y en todos los archivos índice modelo en **SPHINX3/model_architecture/**. Observe que cuando los datos de entrenamiento son escasos, incrementar el número de gaussianas/estado aumenta el número de parámetros que usted debe entrenar, y produce pésimos modelos cuando este número es demasiado alto, dado que los datos disponibles para el entrenamiento pueden ser insuficientes. Para utilizar otros modelos acústicos, modifique simplemente **rm1.props** para apuntar a ellos.

Evaluación del rendimiento: El archivo de registro de decodificación **filename.log**, contiene toda la información que precisa para evaluar el funcionamiento del decodificador para esta práctica. Para cada archivo de rasgos **foo*.mfc** decodificado, encontrará una entrada que será generalmente como la que se muestra a continuación:

```
Decoding: foo1.mfc

REF:      display posits for the hooked track with chart switches set to their default values
HYP:      display posits for the hooked track with chart switches set to their default values
ALIGN_REF: display posits for the hooked track with chart switches set to their default values
ALIGN_HYP: display posits for the hooked track with chart switches set to their default values

Accuracy: 100.000%  Errors: 0 (Sub: 0 Ins: 0 Del: 0)
Words: 14  Matches: 14  WER: 0.000%
Sentences: 1  Matches: 1  SentenceAcc: 100.000%
HypScore: -1.791845E7
This Time Audio: 4.50s  Proc: 17.55s  Speed: 3.90 X real time
Total Time Audio: 4.50s  Proc: 17.55s  Speed: 3.90 X real time
Mem Total: 437.55 Mb Free: 154.72 Mb Used: 282.83 Mb
===== statistics for batch=====
totalStates 242134.0
totalArcs 1065032.0
actualArcs 937912.0
totalTokensScored 1419847.0
curTokensScored 1419847.0
tokensCreated 3381309.0
-----

Decoding: foo2.mfc

REF:      how many ships were in galveston may third
HYP:      how many ships were in galveston dates there
ALIGN_REF: how many ships were in galveston MAY THIRD
ALIGN_HYP: how many ships were in galveston DATES THERE

Accuracy: 90.909%  Errors: 2 (Sub: 2 Ins: 0 Del: 0)
Words: 22  Matches: 20  WER: 9.091%
Sentences: 2  Matches: 1  SentenceAcc: 50.000%
HypScore: -1616598.2  ActScore: NONE
This Time Audio: 2.70s  Proc: 9.46s  Speed: 3.51 X real time
Total Time Audio: 7.19s  Proc: 27.01s  Speed: 3.76 X real time
Mem Total: 437.55 Mb Free: 168.25 Mb Used: 269.30 Mb
===== statistics for batch=====
totalStates 242134.0
totalArcs 1065032.0
actualArcs 937912.0
totalTokensScored 2267396.0
curTokensScored 847549.0
tokensCreated 5393013.0
-----
```

Esta salida corresponde al reconocimiento de dos archivos de rasgos, foo1.mfc y foo2.mfc en secuencia. En el ejemplo de arriba, las líneas que empiezan por REF: y HYP: muestran las secuencias de palabras correctas y planteadas como hipótesis para el archivo respectivamente. Las líneas que comienzan por ALGIN_REF: y ALIGN_HYP: muestran el resultado después de la referencia y las secuencias de palabras decodificadas fueron alineadas mediante un algoritmo de distorsión dinámico-temporal (DTW). Las palabras en mayúsculas eran de algún modo erróneas. En este ejemplo, foo1.mfc fue decodificada correctamente. La entrada **Accuracy** (precisión) indica el porcentaje de palabras del grupo de pruebas que fueron reconocidas correctamente hasta el momento (100% en el momento en que foo1.mfc fue decodificada). Observe de nuevo que esto es una métrica acumulativa, y hace referencia a todos los archivos decodificados hasta el momento. No obstante, no es una métrica suficiente; es posible plantear correctamente como hipótesis todas las palabras de los enunciados de prueba simplemente hipotetizando un gran número de palabras para cada palabra del grupo de prueba. Las palabras falsas, conocidas como inserciones, también deben penalizarse cuando se mida el funcionamiento del sistema. La entrada **WER%** indica el número de palabras hipotéticas que eran erróneas como porcentaje del número real de palabras del grupo de prueba. Esto incluye a las palabras que se hipotetizaron erróneamente (o se eliminaron), y a las palabras que se insertaron falzamente. Dado que el reconecedor puede, en principio, hipotetizar muchas más palabras falsas que palabras existen en el grupo de prueba, el

porcentaje de errores puede realmente ser mayor que 100. Toda la métrica notificada en el archivo de registro, es métrica acumulativa. En el ejemplo anterior, después de que foo2.mfc fue decodificada, de las 22 palabras de la prueba de referencia *hasta el momento*, se hipotetizaron correctamente las transcripciones de 20 palabras (90.909%). En el proceso, el reconocedor hipotetizó 2 palabras falsas (aquí se incluyen inserciones, eliminaciones y sustituciones, que se señalan con las entradas **Ins:**, **Del:** y **Sub:** respectivamente). La línea

```
Sentences: 2 Matches: 1 SentenceAcc: 50.000%
```

indica que 2 oraciones han sido decodificadas hasta el momento, de las cuales, 1 fue correctamente reconocida, dando una precisión en la oración del 50%.

La precisión en el reconocimiento de la oración, es una medida muy importante en tareas en las que es necesario un reconocimiento absolutamente correcto, como es el caso del reconocimiento de identidades o números de tarjetas de crédito, o en máquinas con respuestas de resultado crítico.

Las líneas

```
Tiempo actual del audio: 2.70s Proc: 9.46s Velocidad: 3.51 X tiempo real
Tiempo total del audio: 7.19s Proc: 27.01s Velocidad: 3.76 X tiempo real
Memoria Total: 437.55 Mb Libres: 168.25 Mb Usados: 269.30 Mb
```

se refieren a la velocidad de decodificación y uso de memoria. Desde arriba hacia abajo, leyendo de izquierda a derecha, una interpretación de las líneas anteriores podría ser: "la duración de la oración actual fue de 2.7 segundos, el tiempo de procesado para la misma fue de 9.46 segundos, fue decodificada 3.51 veces en tiempo real; la duración total de los enunciados decodificados hasta el momento fue de 7.19 segundos, el tiempo de procesamiento total fue de 27.01 segundos, la velocidad media fue de 3.76 veces en tiempo real; la memoria total que está siendo utilizada por el JVM es 437.55 Mb, de los cuales 168.25 Mb están aún libres y 269.30 Mb están siendo utilizados". Fíjese en que JVM consume bastante memoria. Utilizará tanta memoria como le sea asignada. El truco consiste en asignar sólo lo suficiente para su tarea, a través de los indicadores -mx y -ms descritos anteriormente.

El estudio de los errores cometidos en el proceso de reconocimiento (las palabras en mayúsculas en la referencia alineada y secuencias de palabras hipotéticas) proporciona normalmente una idea de lo que podría hacerse para mejorar el reconocimiento. Por ejemplo, si la palabra "FOR" se ha hipotetizado como la palabra "FOUR" casi todo el tiempo, quizás sea necesario que corrija la pronunciación de la primera en su diccionario de decodificación e incluya una pronunciación que asocie la palabra FOR con las unidades utilizadas en el mapeo de la palabra FOUR. Una vez realizadas estas correcciones, debe volver a codificar.

Decodificación en vivo Esto está simplemente destinado a demostrar cómo la decodificación en vivo se puede llevar a cabo con los modelos que usted entrenó. Vaya al directorio **sphinx4/tests/live**, asegúrese de que el archivo **rm1_live.props** señala a los modelos acústicos correctos, y utilice el comando **make live** de la línea de comandos. Se

abrirá una interfaz GUI en su pantalla. Elija el botón "*your experiment*" (su experimento) y espere hasta que esté listo. Pulse luego el botón de hablar y pronuncie la oración solicitada por el sistema por el micrófono. Pulse de nuevo el botón de hablar después de que haya acabado de hablar. La salida del reconocedor aparecerá en la ventana de hipótesis de la interfaz GUI. El resultado del reconocimiento será probablemente nefasto. La razón de esto es que un decodificador en vivo debe optimizarse de muchos modos en entornos que sean específicos para una máquina y tarea determinadas. Ese no es el centro de esta práctica. Observe también que si la máquina en la que está ejecutando esto no es físicamente la misma que la máquina en la que usted está hablando, el decodificador en vivo no funcionará. En tal caso, tendrá que copiar el sistema en su máquina local y a continuación, deberá ejecutarlo. Si no le interesa, puede saltarse esta parte.

APÉNDICE 1

Si su archivo de transcripción tiene las siguientes entradas:

THIS CAR THAT CAT (archivo1)
CAT THAT RAT (archivo2)
THESE STARS (archivo3)

y su diccionario del lenguaje tiene las siguientes entradas para estas palabras,

CAT K AE T
CAR K AA R
RAT R AE T
STARS S T AA R S
THIS DH I S
THAT DH AE T
THESE DH IY Z

entonces, la frecuencia de aparición para cada uno de los fonos es como se muestra a continuación (en un escenario real donde esté entrenando modelos de trifonos, también tendrá que contar los trifonos):

K 3 S 3
AE 5 IY 1
T 6 I 1
AA 2 DH 4
R 3 Z 1

Dado que existen sólo ejemplos simples de las unidades de sonido IY y I, y éstas representan sonidos muy parecidos, podemos fusionarlas en una unidad de sonido que representaremos como I_IY. Podemos pensar también en la fusión de las unidades de sonido S y Z, que representan sonidos muy parecidos, dado que sólo existe un caso de la unidad Z. Sin embargo, si fusionamos I y IY, y también S y Z, las palabras THESE y THIS

no se distinguirán. Tendrán la misma pronunciación, como puede observar en el siguiente diccionario de unidades fusionadas:

CAT K AE T
CAR K AA R
RAT R AE T
STARS S_Z T AA R S_Z
THIS DH I_IY S_Z
THAT DH AE T
THESE DH I_IY S_Z

Si es importante en su tarea ser capaz de distinguir entre THIS y THESE, al menos una de estas dos fusiones no debería realizarse.

APÉNDICE 3

Considere la siguiente oración.

CAT THESE RAT THAT

Si utilizamos el primer diccionario facilitado en el apéndice 1, esta oración puede expandirse como la siguiente secuencia de unidades de sonido:

<sil> K AE T DH IY Z R AE T DH AE T <sil>

Los silencios (denotados como <sil>) han sido agregados al principio y al final de la secuencia para indicar que la oración va precedida y seguida por silencio. Esta secuencia de unidades de sonido tiene la siguiente secuencia de trifonos,

K(sil,AE) AE(K,T) T(AE,DH) DH(T,IY) IY(DH,Z) Z(IY,R) R(Z,AE) AE(R,T) T(AE,DH)
DH(T,AE) AE(DH,T)
T(AE,sil)

donde A(B,C) representa un ejemplo del sonido A cuando el sonido que le precede es B y el siguiente sonido es C. Si cada uno de estos trifonos fuese modelado por un HMM individual, el sistema necesitaría 33 únicos estados, que enumeramos así:

K(sil,AE) 0 1 2
AE(K,T) 3 4 5
T(AE,DH) 6 7 8
DH(T,IY) 9 10 11
IY(DH,Z) 12 13 14
Z(IY,R) 15 16 17
R(Z,AE) 18 19 20
AE(R,T) 21 22 23
DH(T,AE) 24 25 26
AE(DH,T) 27 28 29
T(AE,sil) 30 31 32

Aquí, los números que siguen a cualquier trifono representan los índices globales de estados HMM para cada trifono. Observamos aquí, que excepto en el caso del trifono T(AE,DH), todos los demás sólo aparecen una vez en el enunciado. Por tanto, si tuviésemos que modelar todos los trifonos de forma independiente, los 33 estados HMM tendrían que ser entrenados. Observamos aquí que cuando DH va precedido por el fono T, la realización de la porción inicial de DH sería muy parecida, independientemente de que el fono siga a DH. Por tanto, el estado inicial de los trifonos DH(T,IY) y DH(T,AE) puede estar atado. Utilizando una lógica similar, los estados finales de AE(DH,T) y AE(R,T) pueden estar atados. Otros pares como estos también aparecen en este ejemplo. Atar estados utilizando esta lógica modificaría la tabla anterior, quedando de este modo:

K(sil,AE) 0 1 2
 AE(K,T) 3 4 5
 T(AE,DH) 6 7 8
 DH(T,IY) 9 10 11
 IY(DH,Z) 12 13 14
 Z(IY,R) 15 16 17
 R(Z,AE) 18 19 20
 AE(R,T) 21 22 5
 DH(T,AE) 9 23 24
 AE(DH,T) 25 26 5
 T(AE,sil) 6 27 28

Esto reduce el número total de estados HMM para los que las distribuciones deben ser aprendidas, hasta 29. Sin embargo, se pueden conseguir más reducciones. Podríamos observar que la porción inicial de realizaciones del fono AE, cuando el fono precedente es R, es algo parecida a las porciones iniciales del mismo cuando el fono precedente es DH (debido, por ejemplo, a las consideraciones espectrales). Podríamos por tanto, atar los primeros estados de los trifonos AE(DH,T) y AE(R,T). Si utilizamos una lógica parecida, se podrían atar otros estados y la tabla anterior se modificaría, dando como resultado:

K(sil,AE) 0 1 2
 AE(K,T) 3 4 5
 T(AE,DH) 6 7 8
 DH(T,IY) 9 10 11
 IY(DH,Z) 12 13 14
 Z(IY,R) 15 16 17
 R(Z,AE) 18 19 20
 AE(R,T) 21 22 5
 DH(T,AE) 9 23 11
 AE(DH,T) 21 24 5
 T(AE,sil) 6 25 26

Tenemos ahora sólo 27 estados HMM, en vez de los 33 con los que comenzamos. En grupos de datos más extensos con muchos más trifonos, la reducción del número total de

trifonos puede ser espectacular. El atar estados puede reducir el número total de estados HMM en una o dos órdenes de magnitud.

En los ejemplos anteriores, el atar estados se ha llevado a cabo siguiendo criterios puramente fonético-acústicos. Sin embargo, en un sistema típico de reconocimiento de voz basado en HMM como es el caso de SPHINX, el atar estados no se realiza siguiendo reglas fonético-acústicas, sino otros criterios estadísticos y orientado a datos. Es sabido que estos métodos dan muchos mejores resultados de reconocimiento.

APÉNDICE 4

Modelo de lenguaje: Los sistemas de reconocimiento de voz tratan el proceso de reconocimiento como una de las estimaciones máximas a posteriori, donde la secuencia más probable de palabras es estimada, dando la secuencia de vectores característicos para la señal de voz. Matemáticamente, esto se representa como

$$\text{Word1 Word2 Word3 ...} = \underset{\text{argmax}}{Wd1 Wd2 ...} \{P(\text{feature vectors} | Wd1 Wd2 ...) P(Wd1 Wd2 ...)\} \quad (1)$$

donde Word1.Word2... es la secuencia de palabras reconocidas y Wd1.Wd2... es cualquier secuencia de palabras. El argumento a mano derecha de la ecuación 1 posee dos componentes: la probabilidad de los vectores característicos, dada una secuencia de palabras $P(\text{feature vectors} | Wd1 Wd2 ...)$, y la probabilidad de la propia secuencia de palabras, $P(Wd1 Wd2 ...)$. Los HMM facilitan el primer componente. El modelo de lenguaje facilita el segundo componente, también conocido como componente del lenguaje. Los modelos de lenguaje más comúnmente utilizados son modelos de lenguaje N-grama. Estos modelos suponen que la probabilidad de cualquier palabra en una secuencia de palabras sólo depende de las anteriores palabras N de la secuencia. Por tanto, un modelo de lenguaje grama-2 o bigrama computaría

$$P(Wd1 Wd2 ...) \text{ as } P(Wd1 Wd2 Wd3 Wd4 ...) = P(Wd1)P(Wd2|Wd1)P(Wd3|Wd2)P(Wd4|Wd3)... \quad (2)$$

De igual modo, modelo grama-3 o trigramas lo computaría como

$$P(Wd1 Wd2 Wd3 ...) = P(Wd1)P(Wd2|Wd1)P(Wd3|Wd2, Wd1)P(Wd4|Wd3, Wd2) ... \quad (3)$$

El modelo de lenguaje que se utilizará en esta práctica es un modelo de lenguaje bigrama.

Peso del lenguaje: Aunque una estimación estricta máxima a posteriori seguiría a la ecuación (1), en la práctica, la probabilidad del lenguaje se eleva a un exponente para el reconocimiento. Aunque no existe una clara justificación estadística para esto, se explica frecuentemente como el “equilibrio” del lenguaje y de los componentes de probabilidad acústica durante el reconocimiento, y es sabido que juega un papel muy importante para el reconocimiento de voz. Por tanto, la ecuación de reconocimiento sería

$$\text{Word1 Word2 Word3 ...} = \underset{\text{argmax}}{Wd1 Wd2 ...} \{P(\text{feature vectors} | Wd1 Wd2 ...) P(Wd1 Wd2 ...)^\alpha\} \quad (4)$$

Aquí *alpha* es el peso del lenguaje. Los valores óptimos de *alpha* están típicamente entre 6 y 11.