Desarrollo de un sistema de reconocimiento automático del habla

UNIVERSIDAD
POLITECNICA
DE VALENCIA

etsinf

Escuela Técnica
Superior de Ingeniería
Informática

Realizado por:
Marcos Calvo Lance

Dirigido por:
Emilio Sanchis Arnal
Jon Ander Gómez Adrián

10 de septiembre de 2010

Índice general

Capítulo 1

Introducción

1.1. El reconocimiento automático del habla en el ámbito del reconocimiento de formas

El reconocimiento de formas puede definirse, en el marco de la informática, como el área que pretende que los computadores reconozcan su entorno (y posiblemente interactúen con él) a través de señales captadas por sensores conectados al sistema. En este ámbito se sitúa la disciplina del reconocimiento automático del habla, sobre la cual versa el presente trabajo. El objetivo de esta disciplina es que el sistema informático sea capaz de discernir qué ha dicho un cierto locutor y actuar en consecuencia (por ejemplo, en el caso de un sistema de dictado imprimiendo por pantalla la frase reconocida, o en el caso de un sistema de diálogo poniendo en marcha el proceso de inferencia de una respuesta para dicha frase).

Como puede intuirse, el reconocimiento automático del habla proporciona una nueva forma de interactuar con un computador, en este caso a través de la voz, lo que abre un gran abanico de aplicaciones prácticas como por ejemplo lo son:

- Sistemas de dictado, donde lo que se pretende es una transcripción textual lo más exacta posible de aquello que ha dicho un locutor.
- Sistemas de diálogo, donde el objetivo es conceptualizar aquello que se ha captado por el *sensor auditivo* e inferir una respuesta (la cual, para dotar de mayor naturalidad al sistema puede ser devuelta al usuario mediante un sintetizador de voz).

- Sistemas de recuperación de información (*information retrieval* en inglés), los cuales efectúan búsquedas sobre grandes bases de datos de voz intentando recuperar aquellos documentos que traten de un tema en concreto.
- Localización de palabras clave en documentos hablados (o *word spotting* en inglés), donde el objetivo es encontrar todas las frases o documentos de un corpus que contengan una o varias palabras clave concretas.
- Aplicaciones biométricas, como identificación del locutor.
- Traducción simultánea de textos hablados.

Sin embargo, como es bien sabido en el ámbito del reconocimiento de formas, es imposible no cometer nunca ningún error en tareas *reales* de reconocimiento, por lo que nuestro objetivo no será siempre saber qué ha dicho exactamente el locutor, sino cometer el mínimo número posible de fallos procurando mantener el máximo de información relevante para la aplicación.

En definitiva, el reconocimiento automático del habla es un campo con gran interés práctico y que presenta problemas no precisamente triviales de resolver. Este es el terreno en el que se situa este proyecto, cuya descripción y objetivos se exponen a continuación.

1.2. Descripción y objetivos del proyecto

El presente proyecto se centrará en la construcción de un reconocedor automático de habla continua y, en concreto, del algoritmo de búsqueda, que estará basado en el algoritmo de Viterbi. El entrenamiento de los modelos acústicos y de lenguaje quedará fuera de los límites del trabajo, así que se asumirá que éstos forman parte de la entrada al programa desarrollado. También se asume que los datos de entrada ya han sido preprocesados (parametrización de la señal acústica). Sin embargo, sí forma parte del trabajo la construcción de una estructura de datos que soporte el modelo de lenguaje, de forma que su posterior uso por parte del algoritmo de reconocimiento sea lo más eficiente posible. Además, también se contemplará el caso de pronunciaciones alternativas de una misma palabra.

Para solucionar los problemas anteriores se ha llevado a cabo una implementación en lenguaje C del algoritmo de Viterbi dirigida por el modelo de lenguaje (aproximación *top-down*). De este modo, es el modelo de lenguaje el encargado de indicar al modelo acústico-fonético qué palabras debe intentar reconocer, lo cual hace que se acote el número de hipótesis cuando se lleva a cabo un cambio de palabra. Para implementar esta aproximación se ha decidido mantener el

modelo de lenguaje como un autómata obtenido a partir de un algoritmo de inferencia gramatical para lenguajes k-explorables en sentido estricto [5, 17, 19], con el añadido de que se han tenido en cuenta las posibles transiciones por *back-off* [9]. Además, se ha tenido que construir un árbol de transcripciones fonéticas para cada palabra, dado que la solución clásica de representar todo el vocabulario como un único árbol de prefijos no nos sirve para nuestros intereses (en posteriores capítulos se explicará el por qué de esta afirmación). En resumen, en esta aproximación se permiten tanto transcripciones fonéticas alternativas para una misma secuencia ortográfica como modelos de lenguaje basados en n-gramas, no estando dicha n limitada por el propio software de reconocimiento.

El objetivo principal del presente proyecto será, por tanto, implementar un reconocedor de habla que cumpla los requisitos anteriores y produzca el mínimo error posible en reconocimiento. Un requisito adicional será que dicho reconocedor deberá funcionar en tiempo real para vocabularios de tamaño menor de unas 2000 palabras y en menos de 10 veces tiempo real para vocabularios de talla aproximadamente 50000.

Para comprobar la eficacia del reconocedor implementado se experimentará utilizando la base de datos Albayzin [12, 13]. De este modo se comprobará que el reconocedor funciona de acuerdo con los requisitos especificados.

El resto de la presente memoria se estructura como sigue. En la primera parte del capítulo 2 se exponen brevemente algunas reseñas históricas sobre el reconocimiento automática del habla y cuál es su estado del arte actualmente, mientras que en la segunda parte se formaliza este problema, constituyendo así la base para desarrollos posteriores. En los tres siguientes capítulos se hablará de las decisiones tomadas para poder representar las diferentes unidades de información necesarias para el reconocedor, así como de los fundamentos teóricos en que se basan dichas representaciones. El sexto capítulo estará dedicado al algoritmo de búsqueda, donde se formalizará el espacio sobre el que ésta se ejecutará, se expondrá el algoritmo utilizado para llevarla a cabo, así como un heurísitico, y se discutirá la influencia de éstos sobre las estructuras de datos anteriores. Seguidamente, en el capítulo 7, se comentarán los experimentos realizados para comprobar el correcto funcionamiento del reconocedor, así como el cumplimiento de los requisitos especificados en esta introducción. Por último, el capítulo 8 estará dedicado a exponer las conclusiones a las que se ha llegado.

Capítulo 2

El reconocimiento automático del habla

2.1. Estado del arte

Como puede desprenderse de lo expuesto en el capítulo anterior, el reconocimiento automático del habla es una disciplina que queda situada a caballo entre los ámbitos de algunas de las áreas de estudio más importantes de nuestros días, las cuales son, según [10], la informática, por su relación con el reconocimiento de formas, el procesamiento del lenguaje natural, la informática teórica y la algorítmica; y la lingüística y la psicología, dado que lo que se pretende reconocer está íntimamente ligado al proceso lingüístico y psicológico de la comunicación interpersonal.

En el campo que nos ocupa, el tocante a la informática, se ha producido una gran evolución desde que en la década de 1930 se hicieran los primeros intentos para reconocer sonidos o pequeños vocabularios de palabras aisladas [10, 2]. Así, en la década de los 50 se comenzaron a utilizar aproximaciones basadas en fonemas para reconocer vocabularios de dígitos, y poco después el empleo de técnicas generales del reconocimiento de formas tales como la programación dinámica (y en concreto el *Dynamic Time Warping*), el *clustering* o la cuantificación vectorial comenzaron a extenderse. Gracias a esto, en la década de los 80 aparecieron las aproximaciones que hoy en día son justamente las más utilizadas: las basadas en modelos ocultos de Markov y su algoritmo de entrenamiento conocido como Forward-Backward y el algoritmo de Viterbi para reconocimiento [15]. Además, surgieron nuevas aproximaciones para la estimación de las probabilidades en los modelos de Markov tales como el uso de redes neuronales, dando lugar a lo que

hoy conocemos como modelos híbridos. En la actualidad, además de los intentos de muchos investigadores por hallar técnicas nuevas para mejorar las tasas de reconocimiento en este ámbito (por ejemplo, intentando dar cuenta de fenómenos lingüísticos como la ironía y la prosodia), también es un ámbito de profunda investigación la optimización de los recursos hardware para acelerar los procesos relacionados con el reconocimiento automático del habla, por ejemplo haciendo uso de los avances en computación paralela y distribuida.

Por otro lado, dentro de los sistemas de reconocimiento de voz pueden establecerse múltiples divisiones y clasificaciones con respecto a ciertas características. Por ejemplo, podemos decir que un sistema es dependiente del locutor cuando antes de utilizarlo el usuario debe reajustar el sistema dando, a petición de este último, algunas muestras de su voz. Si esto no ocurre, el sistema se denomina independiente del locutor. Otra división interesante es la existente entre reconocimiento de habla continua y de palabras aisladas. Por último, mencionar otras dos taxonomías importantes: las que pueden establecerse según el tamaño del vocabulario de entrada (desde vocabularios de dígitos a conjuntos de más de 50000 palabras) y las concernientes al modelo de lenguaje utilizado (autómatas finitos, modelos sensibles al contexto, etc).

Como se comentó un poco más arriba, el algoritmo de búsqueda más utilizado actualmente en reconocimiento del habla es el conocido como algoritmo de Viterbi, el cual se expondrá con mucho más detalle en capítulos posteriores. Sin embargo, cabe destacar que no es la única técnica utilizada. Otra de las aproximaciones más empleadas por los investigadores hoy en día es la utilización de las distintas fuentes de conocimiento de forma secuencial para resolver el problema de los espacios de búsqueda exageradamente grandes [6]. Esta técnica está basada en la obtención de una representación intermedia mediante un primer algoritmo de búsqueda, en el que suele utilizarse principalmente la información proporcionada por el modelo acústico-fonético y cuyo resultado suele ser un grafo de fonemas o segmentos; y una segunda búsqueda sobre éste empleando la información del modelo de lenguaje, con el objetivo de encontrar las diferentes palabras y, en consecuencia, la frase pronunciada.

Fijándonos en otro aspecto del algoritmo de Viterbi, veremos que éste es un algoritmo síncrono, es decir, que todas las hipótesis que mantiene representan el mismo instante temporal. Es interesante mencionar que también existen algoritmos que no respetan este sincronismo, los cuales están basados en la búsqueda A* [14]. Para una visión más amplia de las técnicas de búsqueda más utilizadas en la actualidad puede consultarse [1].

Hoy en día es tal la importancia del reconocimiento automático del habla que existen numerosas herramientas disponibles (muchas de ellas distribuidas gratuitamente) útiles para el entrenamiento de modelos e incluso para el reconocimiento. Algunas de las más famosas y utilizadas a nivel internacional son los *toolkits* HTK [20] y SPHINX [11] para el entrenamiento de modelos ocultos

de Markov, los cuales son, como ya se ha mencionado, muy utilizados en el modelado acústico, y SRILM [18] y CMU SLM [16] para modelado de lenguaje.

Tal vez la prueba más convincente de la importancia de esta disciplina en la actualidad es la gran cantidad de aplicaciones, ya no sólo en el ámbito universitario, sino también en el comercial, que la utilizan. Algunos ejemplos son [10]:

- Los agentes de diálogo empleados por algunas compañías aéreas como *United Airlines* en sus servicios de atención al viajero.
- Los sistemas controlados por voz ya existentes en muchos automóviles y en la Estación Espacial Internacional.
- Los sistemas de subtitulado automático como el que ya está aplicando *YouTube* a sus vídeos.
- Los sistemas que analizan las opiniones y preferencias de usuarios de ciertos servicios a partir de breves entrevistas con ellos.

En resumen, lo que esta breve enumeración sugiere es que el campo del reconocimiento automático del habla y el procesamiento del lenguaje natural son áreas que están experimentando un gran auge en la actualidad y en cuya investigación y desarrollo merece la pena invertir recursos.

2.2. El problema del reconocimiento automático del habla

Una de las características comunes a la mayor parte de aplicaciones prácticas del reconocimiento automático del habla es que, como primer paso, obtienen una representación basada en palabras o, de forma más general, conceptos, de aquello que se ha dicho para después llevar a cabo el objetivo de la aplicación. Por tanto, una de las definiciones de un reconocedor de habla [9] es la de un programa que transcribe automáticamente aquello dicho por un locutor, de modo que esta transcripción puede simplemente mostrarse por pantalla o utilizarse como entrada para algún otro programa.

Tras definir un reconocedor de habla de esta forma, puede verse fácilmente que de forma natural un computador no posee la información necesaria para poder transcribir aquello dicho por un hablante. Esto implica que, para llevar a cabo nuestro objetivo, al sistema de reconocimiento deberán proporcionársele al menos los siguientes datos:

- Un modelo acústico-fonético donde queden plasmadas las características acústicas de cada uno de los fonemas del lenguaje objetivo. Esto viene motivado por el hecho de que la entrada a nuestro sistema será una señal acústica captada por un sensor (típicamente uno o varios micrófonos).[1]
- Un conjunto de palabras que constituirán el vocabulario que poseerá el reconocedor para intentar efectuar la transcripción.
- El conjunto de transcripciones fonéticas de cada palabra, para poder así efectuar una correspondencia entre el modelo acústico-fonético y el vocabulario.
- Información sobre cómo se relacionan entre sí las palabras del vocabulario, con el objetivo de utilizarla como guía para la transcripción. A este conjunto de relaciones se le denomina modelo de lenguaje.

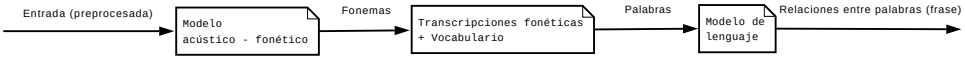

Figura 2.1: Informaciones que necesita un sistema de reconocimiento automático del habla y relaciones entre ellas

Una objeción que puede ponerse a este conjunto de datos es que en el caso de palabras homógrafas (por ejemplo, *bolsa* como recipiente y *bolsa* como institución financiera) es imposible distinguir cuándo queremos expresar cada uno de sus posibles significados. Esto viene motivado por el hecho de las palabras vienen determinadas por su ortografía y tiene como consecuencia que el modelo de lenguaje podría verse penalizado por ello, ya que podrían producirse confusiones entre conceptos. A modo de ejemplo decir que si durante un proceso de reconocimiento se tuviera como hipótesis que el usuario ha dicho *bolsa* y el reconocedor se basara en el modelo de lenguaje para ver qué palabras podrían seguir a ésta, éste le devolvería tanto las relacionadas por el concepto *recipiente* como las correspondientes al concepto *institución financiera*.

Por otro lado, es interesante resaltar que existen tanto reconocedores de habla contínua como de palabras aisladas, ambos de gran utilidad práctica. En un reconocedor de habla contínua, dado que se pretende reconocer una secuencia de palabras con sentido propio, el modelo de lenguaje tiene una gran relevancia ya que orientará la búsqueda a través de las relaciones que éste posee. Sin embargo, si nuestro objetivo fuera programar un reconocedor de habla para palabras aisladas el modelo de lenguaje no tendría mucha importancia práctica si tenemos en cuenta el hecho de que no se pretende dar cuenta de ninguna secuencia de palabras. El resto de elementos de información sí son comunes a

[1] Esta señal acústica suele someterse a un preproceso para no trabajar directamente con ella.

ambos tipos de reconocedores ya que los necesitan para poner en relación la entrada (acústica) con la salida (palabras) (ver Figura 2.1).

Si seguimos una aproximación estadística en la construcción del reconocedor (que, por otra parte, es la más usada hoy en día en reconocimiento automático del habla), el clasificador de Bayes o de mínimo error asociado a la tarea puede formularse del siguiente modo:

$$\hat{W} = \arg \max_{W} \Pr(W|A) \tag{2.1}$$

En esta ecuación W simboliza cualquier combinación de palabras del vocabulario, A la señal acústica de entrada al sistema (preprocesada) y $\hat{W}$ la mejor secuencia de palabras dada dicha señal de entrada o, más formalmente, la secuencia de palabras que maximiza la probabilidad *a posteriori* dada la señal acústica de entrada.

De la fórmula anterior puede llamar la atención que, a primera vista, no parece que se siga la aproximación clásica del reconocimiento de formas en la que a cada elemento se le intenta asignar una etiqueta de clase de entre C clases diferentes intentando cometer el mínimo error posible. Sin embargo, hay que tener en cuenta que, en este caso, las etiquetas de clase son justamente todas las posibles secuencias de palabras (de cualquier longitud) sobre el vocabulario; de este modo, aunque el conjunto de posibles etiquetas de clase es muy grande, se cubren todas las posibles frases que pueda decir el locutor (siempre que éste utilice palabras que formen parte del vocabulario del sistema).

Por otro lado, el utilizar una aproximación estadística nos obliga a que tanto el modelo acústico-fonético como el modelo de lenguaje estén basados en probabilidades. Esta afirmación puede justificarse desarrollando a partir de la fórmula anterior (razonamiento expuesto en [9]). Si aplicamos la regla de Bayes al argumento de la maximización, tenemos que:

$$\Pr(W|A) = \frac{\Pr(W)\Pr(A|W)}{\Pr(A)} \tag{2.2}$$

En este caso, $\Pr(W)$ simboliza la probabilidad de que un hablante diga la frase representada por la secuencia W según el modelo de lenguaje, $\Pr(A|W)$ es la probabilidad de que, dada la secuencia de palabras W la señal acústica observada haya sido A, la cual podemos obtener a partir del modelo acústico-fonético, y $\Pr(A)$ es la probabilidad de que un hablante cualquiera emita la señal acústica A. Esto último es realmente difícil de obtener, aunque por suerte podemos eliminarlo dado que es un valor constante ya que la probabilidad condicional que queremos calcular asume que la señal acústica A es conocida. Por tanto,

combinando las ecuaciones 2.1 y 2.2, tenemos que el clasificador de Bayes para esta tarea puede también escribirse como:

$$\hat{W} = \arg\max_{W} \Pr(W) \Pr(A|W) \tag{2.3}$$

A esta ecuación podemos aplicarle una transformación logarítmica para trabajar con logaritmos de probabilidades en vez de con éstas directamente[2], quedando:

$$\hat{W} = \arg\max_{W} \log \Pr(W) + \log \Pr(A|W) \tag{2.4}$$

En consecuencia, lo que nos queda es una maximización que depende de la probabilidad de la secuencia de palabras de la frase, la cual nos la da el modelo de lenguaje y de la probabilidad de que el hablante haya emitido la señal acústica captada por el sensor asumiendo que la frase que realmente se ha dicho es la que estamos considerando en ese momento, probabilidad que puede obtenerse a partir del modelo acústico-fonético.

Otro aspecto interesante a tener en cuenta sobre las dos fórmulas anteriores es el caso de que se permitan pronunciaciones alternativas de una misma palabra, o, lo que es lo mismo, diversas transcripciones fonéticas para una misma representación ortográfica. Si esto fuera así, la maximización se efectuaría sobre los pares {*transcripción ortográfica, transcripción fonética*}, ya que deberíamos permitir tanto que una misma transcripción ortográfica tuviera asociada varias secuencias de fonemas posibles como que una misma secuencia fonética se correspondiera con palabras ortográficamente distintas (como es el caso de los homófonos, por ejemplo *vaca* y *baca*).

Finalmente, señalar que todas las formulas consideradas hasta ahora definen el clasificador de Bayes para el problema del reconocimiento automático de habla, en otras palabras, como la etiqueta de clase viene definida por la secuencia de palabras que se ha reconocido, lo que se busca es la transcripción perfecta de la señal acústica de entrada, por lo que cometer un solo error de transcripción sería tan malo como fallar en todas y cada una de las palabras de la frase de salida. Sin embargo, existen numerosas aplicaciones en el ámbito del reconocimiento del habla que no necesitan la transcripción *exacta* de lo que dijo el locutor, sino que con una aproximación a ella (es decir, una transcripción en la que el número de errores no sea *demasiado elevado*) sería suficiente. Este es el caso, por ejemplo, del *word spotting* [7], donde lo que realmente nos interesa es encontrar las apariciones de ciertas palabras con un error suficientemente bajo.

[2]El hecho de trabajar con los logaritmos de las probabilidades tiene ciertas consecuencias positivas que se expondrán en capítulos posteriores.

Capítulo 3

El nivel acústico-fonético

Como pudo apreciarse en la Figura 2.1, el nivel acústico-fonético es el módulo del reconocedor encargado de dar cuenta de las características acústicas de cada uno de las unidades fonéticas consideradas, en nuestro caso, fonemas.

Idealmente, la entrada al reconocedor será una señal acústica captada por uno o varios micrófonos. Sin embargo, como en muchas de las aplicaciones del reconocimiento de formas, suele ser necesario un preproceso de la señal para ser tratada por el reconocedor. Para ello, ésta suele muestrearse a intervalos regulares de tiempo y en cada uno se calculan los parámetros que caracterizarán a esa muestra de la señal. En nuestro caso, el muestreo se ha hecho cada 10 ms y como parámetros relevantes se ha optado por utilizar la energía, los 12 primeros coeficientes cepstrales según la escala de Mel, más la primera y segunda derivadas de ellos, lo que da lugar a un total de 39 valores reales por cada una de las muestras de audio. Este conjunto de parámetros suele conocerse como *vector de características* o *frame*. El reconocedor desarrollado asume que la entrada serán ya estos parámetros, por lo que es necesario un módulo previo que se encargue de este preproceso.

Del mismo modo, como el entrenamiento de los modelos acústicos también queda fuera del ámbito de este proyecto, se ha creído conveniente establecer una interfaz de manera que puedan proporcionársele al reconocedor modelos ocultos de Markov entrenados de la manera que el usuario desee, pero siempre que éste implemente una serie de funciones que son invocadas por el programa.

Por todo esto, el resto del presente capítulo se estructura en dos partes. Primero se definirán brevemente los modelos ocultos de Markov y cómo son éstos utilizados en reconocimiento de habla. Por último se explicará la interfaz que un potencial usuario debería cumplir para poder utilizar el reconocedor.

3.1. Modelos Ocultos de Markov

En reconocimiento automático del habla el comportamiento acústico de cada una de las unidades fonéticas suele representarse mediante modelos ocultos de Markov (o HMM, de las siglas en inglés de *Hidden Markov Models*). Las unidades fonéticas consideradas serán los 24 fonemas del castellano más dos fonemas "especiales", a los que se ha dado el nombre de *pausa larga* y *pausa corta*, que modelan silencios de distinta duración. De este modo, tendremos un HMM por cada una de estas unidades fonéticas. Así, si queremos obtener la probabilidad de que una cierta secuencia de *frames* haya sido emitido asumiendo que intentamos reconocer un cierto fonema f deberemos utilizar el modelo de Markov asociado a dicho fonema.

Formalmente, un modelo oculto de Markov M puede ser definido por medio de la tupla $M = (A, S, s_0, F, P, Q)$, donde:

- A es el alfabeto de salida del modelo, es decir, los símbolos que pueden emitirse. En reconocimiento automático del habla este alfabeto está formado por todos los posibles vectores de características de una dimensión determinada.
- S es un conjunto de estados.
- s_0 representa un estado que debe pertenecer a S, y es el estado inicial del modelo. Este estado no será emisor.
- F es un conjunto de estados que debe estar contenido en S y representa los estados finales del modelo. Los estados finales no serán emisores ni partirá ninguna transición desde ellos. En reconocimiento automático del habla suele utilizarse un solo estado final.
- P es un conjunto de distribuciones de probabilidad condicional de transición entre estados $p(s|s')$. Cada una de ellas modela la probabilidad de transitar al estado s si nos encontrábamos en el s'. Al tratarse de distribuciones de probabilidad se cumple que

$$\sum_{s \in S} p(s|s') = 1 \tag{3.1}$$

- Q es un conjunto de distribuciones de probabilidad condicional de emisión $q(a|s)$. Cada una de ellas modela la probabilidad de, encontrándonos en un cierto estado s, emitir el símbolo a perteneciente al alfabeto de salida. En reconocimiento del habla, representará la probabilidad de que un cierto *frame* haya sido emitido asumiendo que nos encontramos en un estado s del modelo. Por ser distribuciones de probabilidad también se cumple

$$\sum_{a \in A} q(a|s) = 1 \tag{3.2}$$

En reconocimiento automático del habla, como el alfabeto abarcaría todo el espacio de vectores de una dimensión determinada (en nuestro caso, con 39 componentes), esto puede darnos una idea de que muy posiblemente las probabilidades de emisión que estemos manejando sean relativamente bajas debido a que el alfabeto tiene talla infinita. Además, como el espacio en el que trabajamos es continuo (en concreto $\mathbb{R}^{39}$), habría que sustituir en la expresión anterior el sumatorio por una integral, quedando

$$\int_{\mathbb{R}^{39}} q(a|s)da = 1 \tag{3.3}$$

Para modelar las distribuciones de probabilidad de emisión de cada uno de los estados del modelo se ha utilizado una mixtura de 32 gaussianas, cada una ponderada por un peso específico.

Es importante señalar el papel de las distribuciones de probabilidad de transición entre estados, las cuales determinan fuertemente dos aspectos importantes durante el reconocimiento de un fonema. Por un lado, marcan la duración mínima que éste puede tener, la cual viene dada por la longitud del camino más corto entre los estados inicial y final multiplicada por la inversa de la frecuencia de muestreo (periodo de muestreo). Por otra parte, cada una de estas distribuciones de probabilidad podría verse como el "precio" que debe "pagarse" por utilizar la distribución de probabilidad de emisión correspondiente a uno de los estados alcanzables desde el actual, como se verá en la ecuación 3.4.

Teniendo en cuenta todo esto, si quisiéramos calcular la probabilidad de que una determinada secuencia de *frames* $x_1...x_n$ de longitud n haya sido emitida asumiendo que hemos recorrido la secuencia de estados $s_0s_1...s_ns_{n+1}$ en el modelo de Markov correspondiente a un cierto fonema f, deberemos utilizar la ecuación 3.4. Hay que aclarar que en la secuencia de estados s_0 debe ser el estado inicial del modelo y s_{n+1} un estado final.

$$\Pr(x_1...x_n|s_0s_1...s_ns_{n+1}) = p(s_1|s_0)\prod_{i=1}^{n} q(x_i|s_i)p(s_{i+1}|s_i) \tag{3.4}$$

En esta ecuación cada uno de los elementos del productorio representa la probabilidad de que el *frame* x_i haya sido emitido en el estado s_i multiplicado por la probabilidad de que se transite del estado s_i al s_{i+1} .

Adicionalmente, si ψ es el conjunto de todos los caminos de longitud $n+2$ en el HMM correspondiente al fonema f tales que empiezan en el estado inicial y acaban en un estado final, podemos calcular la probabilidad total de que la secuencia de *frames* $x_1...x_n$ haya sido emitida como

$$\Pr(x_1...x_n|f) = \sum_{s\in\psi} p(s_1|s_0) \prod_{i=1}^{n} q(x_i|s_i)p(s_{i+1}|s_i) \tag{3.5}$$

Puede demostrarse que en el sumatorio anterior suele haber un camino con mucha más probabilidad que los demás, y es en este hecho justamente en lo que se basa el algoritmo de Viterbi, el cual se expondrá más adelante. Pero es interesante remarcar esto ahora para justificar la importancia de la ecuación 3.4, ya que si asumimos conocido el camino de máxima probabilidad (o tenemos alguna manera de construirlo "sobre la marcha", que es lo que se hace en el algoritmo de Viterbi), podemos por un lado evitar hacer el sumatorio y por el otro trabajar con los logaritmos de las probabilidades dado que no sería necesario hacer sumas de probabilidades. Además, el hecho de trabajar con logaritmos es especialmente importante porque aporta eficiencia al cálculo de la estimación de la probabilidad total (aproximándola por la asociada al camino de máxima probabilidad), dado que así no hay que hacer productos en coma flotante y pueden evitarse errores de cálculo numérico (*underflows* por multiplicaciones de números muy pequeños). Utilizando los logaritmos de las probabilidades la ecuación 3.4 quedaría como sigue:

$$\log \Pr(x_1...x_n|s_0s_1...s_ns_{n+1}) = \log p(s_1|s_0) + \sum_{i=1}^{n} \log q(x_i|s_i) + \log p(s_{i+1}|s_i) \tag{3.6}$$

Debido a las ventajas que se han expuesto, esta es la expresión que se utiliza en el reconocedor implementado para combinar las probabilidades proporcionadas por el modelo acústico-fonético. Como consecuencia, el modelo de lenguaje también tendrá que trabajar con logaritmos de probabilidades, pero de esto se hablará más adelante.

Existe también una taxonomía sobre la topología de los HMM que es interesante comentar. La topología de cualquier modelo oculto de Markov puede clasificarse en uno o más de los siguientes grupos, aunque siempre suele considerarse el más restrictivo:

- Ergódica: no hay ningún tipo de restricción, cualquier estado puede alcanzar a cualquier otro.
- Izquierda-derecha: existe un orden topológico del grafo dirigido subyacente tal que todas las aristas van de izquierda a derecha o buclean sobre algún estado[1].

[1]Se entiende que existe un arco entre dos nodos que representan sendos estados cuando la probabilidad de transitar de un estado al otro es distinta de cero.

- Lineal: cuando el modelo es izquierda-derecha y además los nodos pueden dibujarse en una sola línea recta de forma que todas las aristas o son bucles o van de izquierda a derecha.
- Estrictamente lineal: si el modelo es lineal y de cada nodo sólo parte como mucho un arco formando un bucle a sí mismo y otro a un estado diferente.

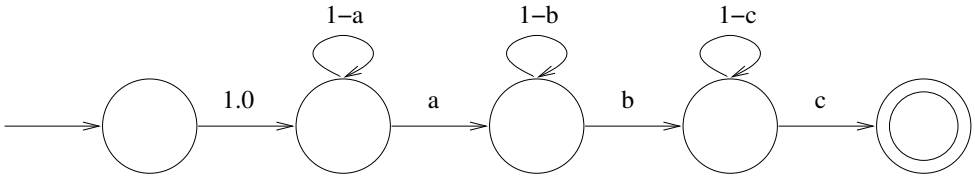

Figura 3.1: Modelo oculto de Markov estrictamente lineal con sus probabilidades de transición asociadas

Hay que aclarar que en la clasificación anterior no debe considerarse el estado inicial, ya que a priori éste podría permitir transitar a cualquiera de los otros estados del modelo sin perjuicio de que éste dejara de considerarse del tipo que el resto de estados indiquen. Por otra parte, a pesar de que en reconocimiento automático del habla suelen utilizarse modelos estrictamente lineales, el reconocedor implementado acepta modelos ergódicos bajo ciertas restricciones de las que se hablará en la sección dedicada a la interfaz para modelos acústico-fonéticos.

Si consideramos todo lo expuesto hasta ahora, podemos plantearnos cómo extender la ecuación 3.6 a palabras y secuencias de palabras teniendo en cuenta la topología de los modelos, aunque lo primero sería ver cómo unir los modelos correspondientes a dos unidades fonéticas. Para hacer esta unión deberemos proceder de la siguiente manera:

Algoritmo 3.1 Método de unión de dos HMM que representan dos unidades fonéticas

1. Lanzamos una transición desde cada estado que esté conectado a algún estado final en el primer modelo hacia todos los estados a los que se pueda llegar en un solo paso desde el estado inicial del segundo modelo. La probabilidad de transición resultante será el producto del sumatorio de las probabilidades de alcanzar un estado final desde el estado origen en el primer modelo por la probabilidad de alcanzar el estado destino de la nueva transición desde el estado inicial del segundo modelo.
2. Eliminamos los estados finales del primer modelo y todas las transiciones que lleguen a ellos.
3. Eliminamos el estado inicial del segundo modelo y todas las transiciones que partan de él.

Una vez sabemos cómo unir dos fonemas es trivial unir todos los de una palabra, e incluso secuencias de palabras. Además, si los modelos involucrados sólo tienen un estado final y el estado inicial sólo alcanza también a un solo estado esta unión se simplifica mucho ya que en el paso 1 del algoritmo anterior hay que añadir únicamente una transición. Es más, como la probabilidad de transición del estado inicial del segundo modelo al destino de la nueva transición es 1, no es necesario hacer el producto, quedando como nueva probabilidad de transición la que estuviera asociada a la que hubiera entre el estado inicial de la nueva transición y el estado final del primer modelo. Como ejemplo de todo esto, en la figura 3.2 se presenta la concatenación de los HMM correspondientes a los fonemas de la palabra *casa* (*/kasa/*).

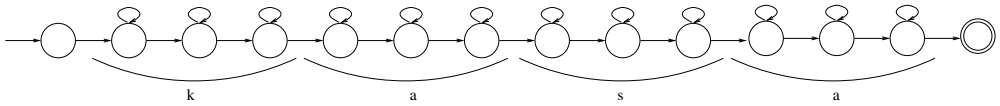

Figura 3.2: Modelo de Markov compuesto para la transcripción fonética */kasa/*

Un detalle que puede apreciarse en esta figura y que se ha tenido en cuenta en el reconocedor implementado es que cuando una misma unidad fonética aparece varias veces en la transcripción, como ocurre en este caso con el fonema */a/*, cada una de ellas debe tratarse por separado y no "reaprovechar" los modelos que ya se hayan concatenado. Si esto se hiciera podrían darse como buenos caminos en el modelo que realmente no lo serían, produciéndose así errores en el reconocimiento.

Por último, si trabajamos con modelos en los que el estado inicial sólo alcanza a un estado, puede eliminarse también el estado inicial del primer fonema ya que la probabilidad de transitar al siguiente estado será siempre 1, quedando como nuevo inicial el estado que se alcanzara desde él. Considerando todo esto, podemos reescribir la ecuación 3.6, donde ahora n representa el número total de *frames* que tiene la entrada y $s_1...s_n s_{n+1}$ es un camino de longitud $n+1$ sobre cualquier concatenación de fonemas tal que forman un número determinado de palabras pertenecientes al vocabulario.

$$\log \Pr(x_1...x_n|s_1...s_n s_{n+1}) = \sum_{i=1}^{n} \log q(x_i|s_i) + \log p(s_{i+1}|s_i) \qquad (3.7)$$

En resumen, partiendo bien de la ecuación 3.6 o de la 3.7, podemos calcular uno de los términos de la ecuación 2.3, en concreto $\Pr(A|W)$. Para ello concatenaremos los modelos de Markov correspondientes a los fonemas de W (recordemos que W representa una transcripción fonética concreta del conjunto de palabras) siguiendo el algoritmo 3.1 y aplicaremos una de dichas fórmulas. A ésta faltaría

aplicarle la información del modelo de lenguaje para tener en cuenta el otro término de la ecuación 2.3, pero de ello se hablará en el capítulo 5.

3.2. Interfaz del reconocedor para modelos acústico-fonéticos

Como ya se ha comentado, se ha decidido establecer una interfaz para que, con independencia de cómo hayan sido entrenados los HMM, éstos puedan ser utilizados por el reconocedor. Esta interfaz está compuesta de las siguientes cabeceras especificadas en lenguaje C:

```
HMMList* HMMList_create (char *fileName);
void HMMList_destroy (HMMList *hmml);

int HMM_get_phoneme_index (HMMList *hmml, char *unit);
int HMM_get_num_states (HMMList *hmml, int phoneme_index);

double* HMM_get_following_states (HMMList *hmml, int phoneme_index,
                                  int from_state_index);
double HMM_pdf (HMMList *hmml, int phoneme_index, int state_index,
                float *frame, int time_stamp);
```

Las funcionalidades que se espera que cada una de ellas posea para permitir que los modelos acústicos puedan ser utilizados correctamente por el reconocedor son las siguientes:

- La función *HMMList_create* debe crear el conjunto de modelos acústicos a partir del fichero cuya ruta se le pasa como parámetro.
- La función *HMMList_destroy* debe liberar la memoria asociada a la estructura de datos que se haya utilizado para mantener los modelos acústicos.
- La función *HMM_get_phoneme_index* debe devolver el identificador entero dentro del modelo acústico-fonético dada la cadena de caracteres por la que se identifica a la unidad fonética en el fichero de correspondencias ortográfico-fonéticas.
- La función *HMM_get_num_states* debe informar del número total de estados (incluyendo no emisores) que tiene el HMM correspondiente a la unidad fonética identificada por el índice que se pasa a la función.

- La función *HMM_get_following_states* debe devolver la probabilidad de transición desde el estado y fonema especificados en la llamada a todos los estados de la misma unidad fonética, incluyendo la probabilidad de buclear sobre él mismo. Esta es la función que permite trabajar con modelos ergódicos en general y no únicamente con modelos izquierda-derecha, dado que se contempla cualquier transición posible entre estados de la misma unidad fonética. Se asume que los índices de los modelos comienzan en 0 (el primer estado emisor es el 1) y que, si el vector devuelto por la llamada es v, $v[i]$ representa la probabilidad de transitar al estado i.
- La función *HMM_pdf* debe proporcionar la probabilidad de que se emita el *frame* que se pasa como parámetro en el estado y unidad fonética especificados. Como último parámetro se puede proporcionar también una marca temporal; brindando la posibilidad de almacenar las probabilidades de emisión ya computadas para un frame determinado de modo que no se repitirían cálculos y se aumentaría así la eficiencia global del reconocedor.

Destacar además que se han impuesto algunas condiciones sobre los modelos que deben respetarse. Por una parte, como ya se comentó, el reconocedor asume que el modelo acústico-fonético trabaja con los logaritmos de las probabilidades. Por otro lado, por razones de simplicidad en la implementación, los modelos de Markov sólo podrán tener un estado final y éste deberá ser el de mayor índice. Asimismo, el estado inicial de todos los modelos sólo podrá estar conectado con el primer estado emisor. Por último, se considera que el valor de -10000.0 asociado al logaritmo de la probabilidad de una transición indica la inexistencia de ésta (como el logaritmo de 0 en cualquier base es $-\infty$, tenemos que asignar a éste un valor numérico para poder hacer comparaciones), debiendo por tanto estar todos los logaritmos de probabilidades distintas de 0 por encima de este número. Queda como trabajo futuro tratar estas tres últimas restricciones, las cuales no han sido eliminadas ya en esta primera versión del reconocedor por el motivo de que en reconocimiento automático del habla suelen trabajarse con modelos estrictamente lineales con dichas características en su topología y se ha considerado que el valor que expresa la inexistencia de una transición entre estados es suficientemente bajo.

Capítulo 4

De fonemas a palabras

Una vez hemos definido el nivel acústico-fonético y hemos visto cómo unir las unidades fonéticas entre sí, nuestro objetivo ahora será intentar encontrar la frase pronunciada por el locutor, para lo que se tratarán de formar secuencias fonéticas correspondientes a palabras del vocabulario atendiendo a las indicaciones que nos proporcione el modelo de lenguaje. Teniendo esto en cuenta, y tal como se vio en la figura 2.1, será necesario disponer del conjunto de posibles transcripciones fonéticas para cada palabra y de un fichero con el modelo de lenguaje. A continuación se expone cómo se ha tratado la primera de estas informaciones en el reconocedor implementado y las ventajas e inconvenientes de las decisiones tomadas.

4.1. Representando las transcripciones fonéticas

Una de las cuestiones que se han tenido que considerar durante la construcción del reconocedor ha sido cómo representar en memoria el conjunto de posibles transcripciones fonéticas para cada palabra. Como soluciones a este problema se consideraron dos aproximaciones, ambas inspiradas en la idea de árbol de prefijos. Sin embargo, antes de exponer y discutir las soluciones consideradas, definiremos el concepto de árbol de prefijos.

En teoría de lenguajes, un árbol de prefijos (también conocido como *trie*) es cualquier autómata finito determinista (AFD) tal que a cada nodo sólo le llega una transición (a excepción del inicial, que no recibe ninguna) y el grafo no dirigido resultante de eliminar las direcciones de las aristas que representan las transiciones no tiene ciclos. De este modo, cada estado del autómata identifica unívocamente sólamente un prefijo de alguna palabra del lenguaje. Un *trie* posee

la interesante propiedad de que puede construirse de forma incremental a partir de las palabras del lenguaje siguiendo el algoritmo 4.1, con un coste $\mathcal{O}(\bar{X} \cdot |L|)$, donde $\bar{X}$ simboliza la longitud media de las cadenas del lenguaje y |L| es el número de palabras del lenguaje. De hecho, estos autómatas tienen la limitación de que sólo pueden dar cuenta de lenguajes finitos. Otra característica que no nos será ventajosa para nuestros fines es que, en general, su número de estados puede llegar a ser relativamente elevado. Si no nos importa perder el hecho de que el autómata sea un árbol de prefijos esto podría solucionarse aplicando algún algoritmo de minimización de AFDs como el basado en el Teorema de Nerode [8] o utilizando en la construcción el algoritmo descrito en [4], el cual obtiene directamente el autómata mínimo a partir de las palabras del lenguaje. El autómata mínimo obtenido a partir de cualquiera de estos dos métodos posee la propiedad de que es un grafo acíclico dirigido.

Algoritmo 4.1 Algoritmo de construcción de un árbol de prefijos a partir de las palabras del lenguaje

---

1. A = autómata mínimo que reconoce $\varnothing$;
2. **Para cada** palabra del lenguaje **hacer**

   Avanzar por A hasta que no se encuentre ninguna transición etiquetada con el símbolo que se está analizando;

   Si se ha analizado toda la palabra

   - Añadir el último estado visitado al conjunto de estados finales de A;

   Si no

   - Añadir a A tantas transiciones y estados como sea necesario para que sea posible dar cuenta del resto de la palabra;
   - Añadir el último estado creado al conjunto de estados finales de A;
3. **Devolver** A;

Una vez definido qué es un árbol de prefijos podemos ver que esta es una buena estructura para mantener las diversas secuencias fonéticas contenidas en el diccionario de correspondencias ortográfico-fonéticas, por un lado porque podría dar cuenta sin problemas del lenguaje compuesto por dichas transcripciones fonéticas ya que existirá siempre un número finito de éstas y por otro porque sería bastante sencillo construirlo siguiendo el algoritmo expuesto. Sin embargo, y dado que cada palabra (representada por su transcripción ortográfica) podría tener más de una transcripción fonética, podemos considerar dos estrategias para mantener en memoria estas correspondencias:

- Construir un solo árbol de prefijos que contemple las transcripciones foné-

ticas de todas las palabras del vocabulario, de modo que los estados finales proporcionen información acerca de las posibles transcripciones ortográficas que hubieran podido originar la secuencia fonética determinada por el camino en el árbol desde la raíz hasta ese nodo.

- Mantener un árbol de prefijos por cada palabra, de forma que se contemplen todas las posibles transcripciones fonéticas para *esa* palabra en concreto.

Es bastante intuitivo ver que el coste espacial de la primera aproximación puede llegar a ser bastante menor que la segunda, ya que, por ejemplo, en el caso de las palabras *pera* y *periódico*, con transcripciones fonéticas */pera/* y */periodiko/* respectivamente, en el primer caso compartirían los estados correspondientes a sus tres primeros fonemas mientras que en el segundo, al estar representados en autómatas diferentes, no compartirían ningún estado. Sin embargo, debido a que se va a utilizar una aproximación dirigida por el modelo de lenguaje (o aproximación *top-down*), necesitamos saber en todo momento qué palabra estamos intentando reconocer. En este caso, la estrategia de construir un único árbol de prefijos no nos es válida, ya que hasta que no alcanzamos un estado final no podemos saber qué palabra hemos estado intentando reconocer, dado que esta información se encuentra en los nodos finales. Por tanto, de las dos posibilidades consideradas, la única que nos es válida para nuestros objetivos si deseamos utilizar una búsqueda *top-down* es aquella que mantiene un árbol de prefijos por cada transcripción ortográfica, ya que en este caso sí podemos saber qué palabra estamos intentando reconocer, lo que nos permitirá aplicar de forma temprana la información proporcionada por el modelo de lenguaje. Como se ha mencionado, esta segunda aproximación tiene un mayor coste espacial que la primera, pero teniendo en cuenta que hoy en día se disponen máquinas con suficiente capacidad esto no debería ser un gran inconveniente si no derrochamos los recursos.

Para representar en memoria esta estructura de datos se ha optado por utilizar un árbol hermano derecho - hijo izquierdo por cada palabra del vocabulario. En este tipo de árbol los sucesores de cualquier nodo están representados como una lista enlazada, así que, como el propio nombre de la estructura de datos indica, cada nodo tiene un puntero al siguiente en la lista de hermanos (hermano derecho) y otro al primero de sus hijos (hijo izquierdo). Además, los árboles hermano derecho - hijo izquierdo tienen la ventaja de que las operaciones que precisamente más nos van a interesar realizar con ellos, que son acceder a los nodos raíz para saber los primeros fonemas de una palabra y a los sucesores de un nodo determinado para conocer los fonemas siguientes a uno dado, se efectúan en tiempo $\mathcal{O}(1)$ simplemente devolviendo el puntero a la raíz o al hijo izquierdo de dicho nodo respectivamente. El procesado de la lista de sucesores puede llevarse a cabo en $\mathcal{O}(n)$ mediante un recorrido lineal utilizando los punteros al hermano derecho, el mínimo tiempo posible si deseamos tener en cuenta todos los elementos de la lista.

En consecuencia, en cada nodo del árbol deberá representarse la siguiente información:

- Identificador del fonema que representa. Utilizaremos para ello, por motivos de eficiencia tanto espacial como temporal, los identificadores enteros que utiliza el modelo acústico-fonético. Cabe destacar que aunque en un AFD cualquiera los símbolos se representarían en las transiciones, si tenemos en cuenta que los árboles de prefijos tienen la propiedad de que el grado de entrada de cada estado es 1 (a excepción del inicial que es 0), podemos representar el símbolo en el propio nodo. De este modo puede obviarse el estado inicial del autómata y tener una lista de estados iniciales (que serían los sucesores de éste), asociando a cada uno el símbolo correspondiente a la transición del estado inicial a él.

- Un indicador que informe de si el estado es final o no.

- Un puntero al hermano derecho y otro al hijo izquierdo.

Para ilustrar lo expuesto en esta sección, en la figura 4.1 puede verse el árbol de prefijos que resultaría si una cierta palabra tuviera asociado el conjunto de transcripciones fonéticas */a/*, */abc/* y */aa/*. En la parte superior de cada nodo se representa la unidad fonética como carácter (aunque en implementación sería un número entero) y en la inferior el elemento de la izquierda debe interpretarse como el puntero al hijo izquierdo y el de la derecha como el puntero al hermano derecho. Un doble borde alrededor del nodo indica que el estado es final.

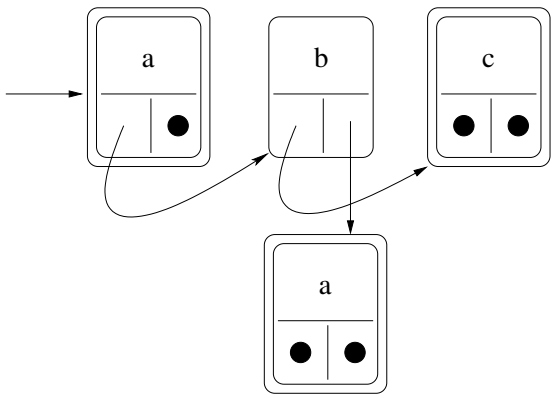

Figura 4.1: Árbol fonético para el caso de que una palabra tenga el conjunto de transcripciones fonéticas {*/a/*, */abc/*, */aa/*}

4.2. El problema de los silencios

Llegados a este punto del diseño de las estructuras de datos del reconocedor podemos plantearnos un problema de gran importancia dentro del reconocimiento automático del habla, relacionado con el fenómeno psicolingüístico de la producción de una frase. Es bien sabido que cuando una persona está hablando puede efectuar pausas entre las palabras o no, lo que en el proceso de reconocimiento implica que cuando se haya terminado de dar cuenta de una cierta palabra debe darse la posibilidad al programa de reconocer un silencio, que puede ser más o menos largo, permitiendo además que, si el locutor ha seguido hablando tras la pausa, las palabras que haya dicho puedan ser reconocidas. A modo de ejemplo, podemos ver que en la frase *"Francia, Suiza y Hungría ya hicieron causa común"*, contenida en el corpus Albayzin [12, 13], es muy probable que un locutor haga una pequeña pausa entre las palabras *Francia* y *Suiza*, pero posiblemente concatenará las palabras *y* y *Hungría*, dando lugar a una pronunciación bastante parecida a */yungria/*.

Otro problema relacionado con este pero de naturaleza ligeramente distinta es que, como aquello que se desea reconocer será captado por uno o varios micrófonos, éstos deberán ser activados antes de que el locutor comience a hablar, y desactivados después de que éste termine. Esto implica que posiblemente se captarán algunos instantes de silencio tanto antes como después de la primera y última palabras pronunciadas respectivamente y el reconocedor debería poder tratar con su posible existencia.

Una solución a estos dos problemas consiste en la introducción del silencio como una palabra especial dentro del modelo de lenguaje, permitiendo que tanto antes de la primera palabra como después de cada una de ellas pueda haber o no un silencio. El árbol de transcripciones fonéticas de esta palabra especial se construiría a partir de todas las unidades correspondientes a silencios que se consideraran en el modelo acústico-fonético[1].

Sin embargo, esta aproximación presenta un problema y es que esta palabra especial *silencio* no debería tenerse en cuenta a la hora de pedirle información al modelo de lenguaje sobre las relaciones entre palabras. Este hecho resulta especialmente molesto cuando se utiliza una búsqueda dirigida por el modelo de lenguaje, como es nuestro caso, por lo que la solución expuesta no se ha considerado adecuada.

Otra forma de solucionar el problema de los silencios, que es la que finalmente se ha implementado en el reconocedor, consiste en considerar que todas las transcripciones fonéticas tienen una serie de prefijos opcionales compuestos por cada una de las unidades fonéticas correspondientes a silencios. De este modo,

[1] Realmente el árbol de prefijos consistiría en una sola lista enlazada compuesta por todas las unidades fonéticas correspondientes a silencios, donde todos los nodos serían finales.

cualquier palabra podrá comenzar tanto por su primera unidad fonética "real" (por ejemplo, en *casa*, el fonema */k/*) como por cualquiera de los silencios, teniendo en cuenta que después de cada uno de los silencios deberán intentar reconocerse las primeras unidades fonéticas de la palabra correspondiente.

Una de las ventajas de esta aproximación es que puede implementarse modificando la idea de árbol hijo izquierdo - hermano derecho expuesta en la sección anterior simplemente creando un preludio en el que se encuentren las unidades fonéticas correspondientes a silencios (algoritmo 4.2). Obviamente, ninguno de los nodos que representen silencios será final.

Algoritmo 4.2 Modificación del árbol de prefijos para dar cuenta de unidades fonéticas correspondientes a silencios

---

Entrada: Árbol de prefijos implementado según la estructura hijo izquierdo - hermano derecho

1. Crear una lista enlazada por medio del puntero "hermano derecho" en la que se hallen todas las unidades fonéticas correspondientes a silencios.
2. Hacer que el puntero "hermano derecho" del último nodo de la lista apunte a la raíz del árbol.
3. Hacer que los punteros "hijo izquierdo" de todos los elementos de la lista apunten a la raíz del árbol.
4. Hacer que la nueva raíz del árbol sea el primer elemento de la lista de silencios.

Mediante la aplicación de estas modificaciones en cada una de las palabras conseguimos que, por una parte, cuando al preguntar por las primeras unidades fonéticas de una palabra se devuelva la raíz del árbol, la lista resultante contenga tanto todos los silencios como los primeros fonemas "reales", por lo que se cumple la opcionalidad de la existencia de los silencios. Por otro lado, cuando se termine de reconocer un silencio y se pregunte por sus siguientes fonemas se devolverá el destino de su puntero al hijo izquierdo, dando como resultado la lista de primeros fonemas "reales" de la palabra en cuestión. Por tanto, consiguen integrarse los silencios dentro del árbol de transcripciones fonéticas de la palabra sin incrementar el coste asintótico de ninguna de las operaciones sobre dicho árbol.

Para clarificar todos los cambios expuestos, a continuación se presenta gráficamente cómo se modificaría el árbol de la figura 4.1 para dar cuenta de las unidades fonéticas *SIL* y *sp*, las cuales representan, respectivamente, un silencio largo y uno corto (*sp* viene de las siglas en inglés de *short pause*).

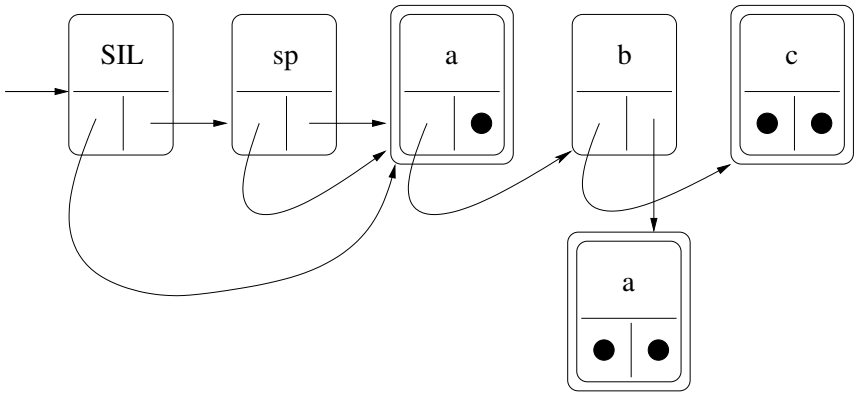

Figura 4.2: Árbol fonético con el preludio de silencios (*SIL* y *sp*) para el caso de que una palabra tenga el conjunto de transcripciones fonéticas {*/a/*, */abc/*, */aa/*}

Recapitulando, la mayor ventaja que se obtiene al aplicar esta estrategia es que consigue integrarse el reconocimiento de los silencios en el caso general del reconocimiento de palabras y fonemas, evitando así crear un caso especial para él. Además, se ha conseguido que todas las operaciones sobre el árbol de fonemas sigan efectuándose en $\mathcal{O}(1)$, excepto el recorrido de listas, que ha de hacerse en $\mathcal{O}(n)$. El único caso especial que debe considerarse es que si al terminar el reconocimiento la hipótesis que devolvería el algoritmo de búsqueda corresponde a un silencio debe descartarse la palabra en la que se encuentre éste y la transición que se haya efectuado en el modelo de lenguaje, ya que no se habrá reconocido ningún fonema *real* de dicha palabra.

Sin embargo, la solución escogida también tiene inconvenientes. El primero es que se agrava todavía más el problema de que el hecho de trabajar con árboles de prefijos es ineficiente en lo que a memoria se refiere, dado que se está replicando la información relativa a silencios en cada una de las palabras. De todos modos, teniendo en cuenta que no suele considerarse un número elevado de silencios, consideramos que el coste extra en memoria de la lista de silencios no es demasiado relevante. Por otro lado, el hecho de tener un preludio de silencios en cada uno de los árboles podría hacer aumentar el número de hipótesis activas en cada iteración del algoritmo de Viterbi, ya que cada vez que se inicie una palabra nueva se incrementará el número de hipótesis creadas en tantas como silencios se hayan considerado. Posiblemente este sí sea el mayor precio que hay que pagar por utilizar esta aproximación, pero, como se verá en los resultados experimentales, a pesar de este aumento en el número de hipótesis se consigue una velocidad en reconocimiento *off-line* de aproximadamente la mitad de tiempo real en las pruebas realizadas con el corpus Albayzin.

Por último, comentar que queda como trabajo futuro la aplicación de alguno de

los algoritmos de minimización expuestos en la sección anterior, con el objetivo de reducir el coste espacial y posiblemente aumentar la eficiencia global del reconocedor dado que se reduciría el número de hipótesis consideradas durante el algoritmo de Viterbi. Además, debería estudiarse cómo afectaría esta minimización al tratamiento de los silencios y si sería necesaria alguna modificación adicional en lo relativo a este aspecto.

Capítulo 5

El modelo de lenguaje

Si recordamos las ecuaciones 2.3 y 2.4, vemos que éstas constaban de dos factores: uno que tiene que ver con la probabilidad de que la acústica captada por los sensores de entrada pueda corresponderse con una secuencia de palabras determinada y otro que es justamente la probabilidad de que pueda darse dicha secuencia. En el capítulo 3 se expuso cómo calcular el primero de ellos mediante una estrategia basada en Modelos Ocultos de Markov. En este capítulo intentaremos ver cómo calcular el segundo factor.

Para conseguir este objetivo, la entrada al reconocedor será un fichero que condendrá un modelo basado en n-gramas en formato ARPA. La representación en memoria de éste se ha llevado a cabo por medio de un autómata que modela un lenguaje k-explorable en sentido estricto, con el añadido de que se han almacenado también las probabilidades de cada uno de los n-gramas. También se han tenido en cuenta las posibles transiciones por *back-off* y las penalizaciones asociadas. Por último, señalar que los efectos del *Grammar Scale Factor* han sido aplicados durante la carga de este modelo. A continuación se desarrollará con más profundidad lo expuesto en este párrafo, detallando cada una de las decisiones tomadas durante la implementación del reconocedor.

5.1. Modelos de lenguaje basados en n-gramas

Como acaba de comentarse, nuestro objetivo será intentar calcular, o mejor dicho, aproximar el valor del término de la ecuación 2.3 que hace referencia al modelo de lenguaje, el cual es $\Pr(W)$. Teniendo en cuenta lo expuesto en capítulos anteriores, W simboliza una secuencia de pares *{transcripción ortográfica,*

transcripción fonética}, secuencia que además de informarnos de la frase pronunciada da cuenta de la transcripción concreta de cada una de las palabras de ésta. Sin embargo, para calcular esta probabilidad sólo nos fijaremos en las transcripciones ortográficas y obviaremos las fonéticas, de las cuales se encargará el modelo acústico-fonético.

Si definimos $W = w_1...w_m$ tenemos que [9]:

$$\Pr(W) = \prod_{i=1}^{m} \Pr(w_i|w_1...w_{i-1}) \tag{5.1}$$

Podemos aproximar esta ecuación asumiendo que sólamente las n palabras anteriores son relevantes para el cálculo de la probabilidad condicional, por lo que quedaría:

$$\Pr(W) \approx \prod_{i=1}^{n-1} \Pr(w_i|w_1...w_{i-1}) \cdot \prod_{i=n}^{m} \Pr(w_i|w_{i-n+1}...w_{i-1}) \tag{5.2}$$

En este caso, para los primeros $n-1$ elementos de la secuencia de palabras W se aplica el primer término de la ecuación, que considera una historia de tamaño menor que $n-1$, es decir, todas las palabras anteriores. Para los subsiguientes elementos se efectúan los cálculos del segundo productorio, el cual tiene siempre en cuenta una historia de $n-1$ palabras, "olvidando" la más antigua cuando se analiza el siguiente elemento de la secuencia. Obviamente, si la secuencia tiene menos de n elementos siempre se aplica el primer factor.

A esta aproximación se la denomina modelo de n-gramas por el hecho de que cuando la secuencia es suficientemente larga (tamaño mayor o igual que n) sólo se toman en consideración las últimas n palabras contando la actual, asumiendo que el resto no influyen en el cálculo de las probabilidades.

Definimos un i-grama (con $i \in \mathbb{N}$) como una secuencia de i palabras que aparecen consecutivas en alguna de las muestras de aprendizaje del modelo de lenguaje[1]. A la vista de la ecuación 5.2 podemos deducir que en un modelo de n-gramas será necesario considerar todos los i-gramas con $1 \leq i \leq n$. Además, teniendo en cuenta que la última palabra de cada secuencia se considera la más nueva, podemos tomar como historia las $i-1$ primeras palabras para calcular las probabilidades de dicha ecuación.

Una forma de representar un modelo de n-gramas es mediante un conjunto de n matrices de talla $|V|^i \ \forall i \in [1, n]$, donde $|V|$ representa la cardinalidad del

[1] El aprendizaje del modelo de lenguaje se efectúa a partir de un conjunto de textos que se consideran representativos del ámbito para el que quiere construirse el reconocedor.

vocabulario y cada uno de los elementos de las distintas matrices de talla i nos informa de la probabilidad estimada de que el i-grama correspondiente pueda darse. Desde el punto de vista de la implementación podemos apreciar que esta idea asume que podemos acceder a las matrices direccionando por palabra, o que existe algún tipo de correspondencia de palabra a entero si el acceso a las matrices es mediante estos últimos, pero este hecho no constituye ningún problema porque existen lenguajes de programación que permiten el primer tipo de acceso y, en su defecto, puede utilizarse alguna técnica de *mapping*. Sin embargo, lo que sí es un problema es el altísimo coste espacial que conlleva este tipo de representación, que es $\mathcal{O}(|V|^n)$. Por ejemplo, si asumimos un vocabulario de 10000 palabras y un modelo de lenguaje basado en trigramas y consideramos que el único contenido de cada elemento de la matriz será un número real de 4 bytes tenemos que el espacio total que necesitaríamos para el conjunto de matrices es superior a 4TB, lo que obviamente resulta exagerado.

Sin embargo, si analizamos un poco más la forma de calcular las probabilidades de los i-gramas, podemos intuir que muchas de ellas serán cero, bien porque son construcciones inválidas en el lenguaje que se quiere modelar (como por ejemplo en castellano *"Una lo avión"*) o bien porque simplemente no aparecían en el corpus de entrenamiento considerado (en el mejor de los casos porque no son relevantes para el ámbito objetivo del reconocedor). Aprovechándonos de este hecho, podemos utilizar técnicas de almacenamiento de datos dispersos que hacen mucho más eficiente la representación en memoria de estas probabilidades. En nuestro caso, se ha optado por construir un autómata utilizando un algoritmo de inferencia gramatical para lenguajes k-explorables en sentido estricto añadiendo probabilidades asociadas a las transiciones, como se verá en la sección 5.2. De todos modos, es interesante señalar que en el caso peor el almacenamiento del modelo de lenguaje sigue siendo un problema exponencial ya que si todos los i-gramas tuvieran probabilidad distinta de cero inevitablemente deberíamos guardarlos todos, lo que sería asintóticamente equivalente al caso anterior del conjunto de matrices.

Uno de los formatos existentes para proporcionar modelos de lenguaje basados en n-gramas como entrada a aplicaciones que los utilicen, y que es precisamente el que esperará el reconocedor implementado, es el denominado ARPA [18]. Este formato consiste en un fichero de texto con una cabecera en la que se informa de cuántos i-gramas de cada tipo hay, seguida de n listas cada una de las cuales contiene los i-gramas para un valor determinado de i empezando por 1 y terminando por n. Las $n-1$ primeras listas consistirán en una serie de líneas cada una de ellas con formato *<logaritmo de la probabilidad> <i-grama> <penalización por back-off>*, mientras que en las líneas de la última lista no se encuentra el último campo. Esta diferencia de formato quedará más clara en la sección 5.3, cuando se hable más en profundidad de las transiciones por *back-off*.

Teniendo en cuenta lo que acaba de exponerse, el hecho de trabajar con los logaritmos de las probabilidades implica que debe modificarse la ecuación 5.2 para tener en cuenta este hecho, quedando:

$$\log \Pr(W) \approx \sum_{i=1}^{n-1} \log \Pr(w_i|w_1...w_{i-1}) + \sum_{i=n}^{m} \log \Pr(w_i|w_{i-n+1}...w_{i-1}) \quad (5.3)$$

Por otro lado, considerando ahora también el modelo acústico-fonético, tal y como aparece en la ecuación 2.4, podemos pensar que, al haber estimado ambos modelos de forma independiente, quepa la posibilidad de que las probabilidades que obtengamos a partir de uno de ellos sean demasiado grandes con respecto a las del otro, de forma que unas probabilidades lleguen a anular la influencia de las otras. Para evitar este hecho suele añadirse un factor multiplicativo constante que afecta al logaritmo de la probabilidad del modelo de lenguaje, aumentando o disminuyendo su relevancia con el objetivo de compatibilizar los valores obtenidos a partir de cada uno de los dos modelos. Llamando α a este factor tenemos que la ecuación 2.4 quedaría como:

$$\hat{W} = \arg \max_{W} \alpha \cdot \log \Pr(W) + \log \Pr(A|W) \quad (5.4)$$

Debido al hecho de que α sirve como factor de escala entre los logaritmos de las probabilidades del modelo de lenguaje y el acústico-fonético, a esta constante suele conocérsela con el nombre de *Grammar Scale Factor*. En la práctica, este parámetro es uno de los valores que deben ajustarse empíricamente en un reconocedor para optimizar su rendimiento.

A continuación, se expondrá cómo se han tratado en la implementación los conceptos expuestos en esta sección para intentar obtener el menor coste tanto espacial como temporal en la representación del modelo de lenguaje y sus operaciones asociadas.

5.2. n-gramas y lenguajes k-explorables en sentido estricto

Como ya se ha dicho a lo largo de este capítulo, la construcción de la representación en memoria del modelo de lenguaje basado en n-gramas se ha efectuado utilizando un algoritmo de inferencia de autómatas para lenguajes k-explorables en sentido estricto. Esta aproximación para representar el modelo de lenguaje ya ha sido probada con éxito previamente, como puede verse en [19]. Siguiendo lo expuesto en [5, 17], la clase de lenguajes k-explorables en sentido estricto puede definirse como sigue con $k \geq 2$:

Sea Σ un alfabeto y $S \subseteq \Sigma^*$ un conjunto de palabras definidas sobre dicho alfabeto. Definimos $\Sigma(S)$ como el conjunto de símbolos de Σ que aparecen en las palabras de S. Además, definimos los siguientes conjuntos:

- El conjunto de prefijos permitidos $I_k(S)$ formado por todos los prefijos de longitud $k-1$ definidos sobre las palabras de S, junto con todas aquellas palabras de este conjunto que no lleguen a dicha longitud. Formalmente,

  $$I_k(S) = \{u \in \Sigma(S)^* : \exists v \in \Sigma(S)^*, uv \in S, |u| = k-1, v \in \Sigma(S)^*\} \cup \{x \in S : |x| < k-1\}$$

- El conjunto de sufijos permitidos $F_k(S)$, definido de forma análoga al anterior pero considerando los sufijos en vez de los prefijos. Igual que en el caso anterior, cuando la longitud de la palabra sea menor que $k-1$ la palabra entera formará parte del conjunto. Equivalentemente,

  $$F_k(S) = \{v \in \Sigma(S)^* : \exists v \in \Sigma(S)^*, uv \in S, |v| = k-1, u \in \Sigma(S)^*\} \cup \{x \in S : |x| < k-1\}$$

- El conjunto de segmentos de longitud k permitidos en las palabras del lenguaje $T_k(S)$. Este conjunto se obtiene tomando todos los segmentos de dicha longitud que aparezcan en S. Utilizando notación matemática,

  $$T_k(S) = \{v \in \Sigma(S)^* : \exists u, w \in \Sigma(S)^*, uvw \in S, |v| = k\}$$

Una vez definidos estos conjuntos, podemos calcular también el conjunto de segmentos prohibidos $T'_k(S)$ del siguiente modo:

$$T'_k(S) = \Sigma^k - T_k(S)$$

Ahora ya estamos en condiciones de caracterizar los lenguajes k-explorables en sentido estricto (también llamados k-TSS por las siglas en inglés de *k-Testable Strict Sense*) con $k \geq 2$ de la siguiente manera[2]:

$$L \text{ es } k\text{-TSS} \iff \exists S \subseteq \Sigma^* : L \subseteq I_k(S)\Sigma^* \cap \Sigma^* F_k(S) - \Sigma^* T'_k(S)\Sigma^*$$

Tras definir la familia de lenguajes k-explorables en sentido estricto, veremos ahora cómo podemos aprovecharnos de estos lenguajes para representar el modelo de n-gramas. Para ello, se verá que el lenguaje formado por todas las frases que pueden obtenerse considerando un modelo de lenguaje basado en n-gramas constituye justamente un lenguaje n-TSS. Primero, consideraremos que el alfabeto Σ está formado por todas y cada una de las palabras que aparecen en el modelo de lenguaje, de forma que consideraremos cada palabra como si de un único símbolo se tratase. Ahora, si tomamos todo el contenido de este modelo como el conjunto de muestras de aprendizaje S tenemos que[3]:

[2] Más adelante se verá cómo se ha tratado el caso de $k = 1$.

[3] Nótese que como se ha hecho $k = n$, los subíndices de los conjuntos ya no son k, sino n.

- $I_n(S)$ estaría formado por todos los i-gramas con $1 \leq i \leq n-1$. Esto es así porque, por un lado estas son todas las secuencias de longitud menor que n, que por definición están contenidas en el conjunto, y por el otro el prefijo de longitud $n-1$ de cualquier n-grama será una secuencia que ya habrá aparecido en el modelo, de acuerdo con el proceso de construcción de éste que se explicó en la sección anterior.

- $F_n(S) = I_n(S)$ por un razonamiento análogo al que se acaba de exponer, cambiando la búsqueda de prefijos por la de sufijos.

- $T_n(S)$ contiene únicamente los n-gramas, es decir, los i-gramas con mayor valor de i. Esto es debido a que los únicos segmentos de longitud n son justamente las secuencias completas de dicha longitud.

Definiendo de esta forma los conjuntos puede verse que, según la caracterización anterior, el lenguaje formado por todas las frases que pueden formarse siguiendo el modelo de n-gramas es un lenguaje n-explorable en sentido estricto. Este lenguaje estaría formado por todas las secuencias contenidas en el modelo de lenguaje más aquellas más largas tales que todos sus segmentos de longitud n pertenecen a $T_n(S)$, hecho coherente además con el cálculo de probabilidades que se trató en la sección anterior, ya que si esta condición no se diera no habría forma de calcular la probabilidad asociada a esa secuencia, aunque más adelante volveremos a hablar sobre este tema. No es necesario indicar que estas secuencias más largas tengan inicios y finales contenidos en los respectivos conjuntos porque, por el propio proceso de construcción del modelo de n-gramas, esto queda garantizado si se cumple la condición relativa a sus segmentos.

Los lenguajes k-TSS constituyen una subclase de los lenguajes regulares, por lo que pueden ser representados mediante un autómata finito determinista (AFD). De hecho, existe un algoritmo de inferencia gramatical capaz de construir un AFD para un lenguaje de esta clase a partir de un subconjunto S de palabras de éste, que es en el que nos basaremos para representar el modelo de lenguaje. A continuación se presenta dicho algoritmo y, a modo de ejemplo, el autómata obtenido con éste para el conjunto de muestras $\{a, ab, abba, bab, babab\}$ con $k = 3$.

A la vista de la figura 5.1, podemos dar una interpretación de los estados y las transiciones del autómata que nos serán útiles para el cálculo de las probabilidades y las operaciones que luego implementaremos sobre él. Considerando primero los estados, vemos que cada uno de ellos tiene asociada una "etiqueta" que indica la historia con la que se ha llegado a él, teniendo en cuenta como mucho las $n-1$ palabras anteriores, lo que se corresponde con las partes derechas de las probabilidades de la ecuación 5.3. Por otro lado, si vemos el autómata como generador de palabras del lenguaje en vez de como aceptor de éstas, es decir, si lo tomamos como si de una gramática regular se tratase, las transiciones pueden interpretarse como los símbolos terminales (o del alfabeto del autómata)

Algoritmo 5.1 Método para la obtención de un AFD A para el menor lenguaje k-TSS que contiene al conjunto de muestras de aprendizaje S [5, 17]

Nota: El formato de las transiciones en este algoritmo será: {*estado origen, símbolo, estado destino*}.

1. Construir los conjuntos $\Sigma(S)$, $I_k(S)$, $T_k(S)$ y $F_k(S)$ a partir del conjunto de muestras de aprendizaje S;

2. Inicialmente, el conjunto de estados del autómata tendrá sólo uno que representará la cadena λ ($Q = \{\lambda\}$), el conjunto de transiciones estará vacío ($\delta = \varnothing$) y el estado inicial será el único que hasta ahora hay en Q ($q_0 = \lambda$). Como puede intuirse, cada estado representará una subcadena válida en el lenguaje;

3. **Para cada** elemento de $I_k(S)$ $a_1a_2...a_m$ **hacer**

   a) $Q = Q \cup \{a_1\}$; /* *Unimos el estado correspondiente al primer símbolo de la cadena a los que ya existieran* */

   b) $\delta = \delta \cup \{\{\lambda, a_1, a_1\}\}$; /* *Desde el estado inicial podemos ir al primer símbolo de la cadena* */

   c) **Para** $j = 1$ **hasta** m **hacer**

      1) $Q = Q \cup \{a_1...a_j\}$; /* *Si no estaba en el conjunto de estados, añadimos el correspondiente al prefijo de longitud j del elemento que estamos considerando* */

      2) $\delta = \delta \cup \{\{a_1...a_{j-1}, a_j, a_1...a_j\}\}$; /* *Añadimos al conjunto de transiciones la que nos permite dar cuenta del prefijo de longitud j de la palabra actual si no formaba parte de él previamente* */

4. **Para cada** elemento de $T_k(S)$ $a_1...a_k$ **hacer**

   a) $Q = Q \cup \{a_1...a_{k-1}, a_2...a_k\}$; /* *Si es necesario, creamos los estados para dar cuenta de la transición entre dos segmentos de longitud k que han aparecido en alguna de las muestras de aprendizaje* */

   b) $\delta = \delta \cup \{\{a_1...a_{k-1}, a_k, a_2...a_k\}\}$; /* *Incluímos además dicha transición en el conjunto de transiciones si no existía previamente* */

5. **Devolver** $A = \{Q, \Sigma(S), \delta, q_0, F_k(S)\}$; /* *Antes de devolver el autómata deberemos marcar como finales los estados correspondientes a los elementos del conjunto de sufijos permitidos* */

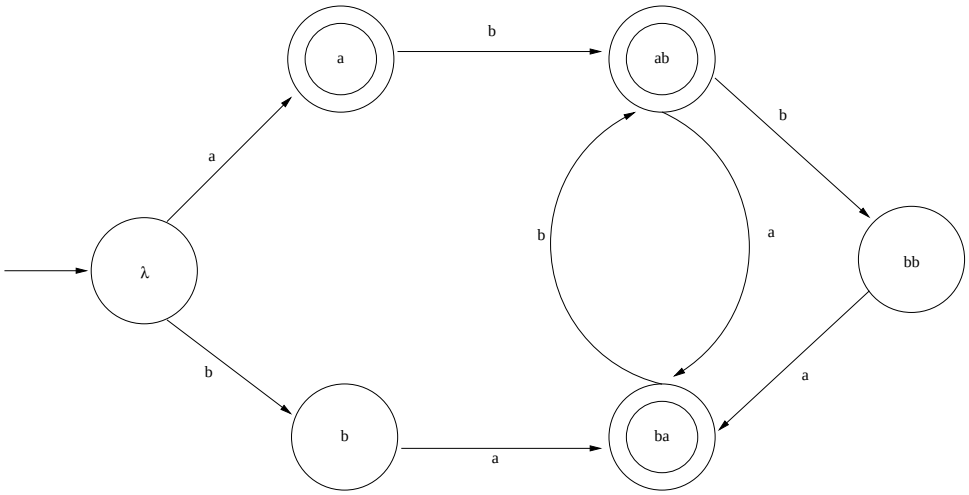

Figura 5.1: Autómata resultante de aplicar el algoritmo 5.1 al conjunto de muestras $S = \{a, ab, abba, bab, babab\}$

que pueden seguir al no terminal determinado por el estado del autómata en el que nos encontremos, además de identificar el estado al que se iría a parar. Este hecho es semánticamente equivalente a las partes izquierdas de las probabilidades condicionales de la mencionada ecuación, por lo que podemos concluir que las probabilidades de transición (o sus logaritmos) podrán guardarse en las transiciones, junto con los símbolos asociados a éstas. De este modo podemos ordenar las transiciones que salen de un determinado estado (que habrán sido obtenidas durante la lectura del fichero ARPA) de mayor a menor probabilidad, que es como se ha hecho en el reconocedor implementado. Esto permitirá que cuando en el proceso de reconocimiento se le pregunte al modelo de lenguaje por las palabras que pueden seguir a un estado dado posiblemente la respuesta no contenga todas las transiciones posibles desde dicho estado, si se da el caso de que alguna de ellas no supera un determinado umbral. Esta forma de proceder será comentada en mayor profundidad en el apartado 6.3.1. Sin embargo, sí es interesante destacar ahora el hecho de que las transiciones salientes de cada estado inducen una distribución de probabilidad sobre las palabras del vocabulario, de forma que se indica la propensión a que después de un cierto i-grama aparezca otra determinada palabra. De este modo, por ser una distribución de probabilidad, para todo estado Q se cumple que

$$\sum_{w \in \Sigma} \Pr(\delta(Q, w)) = 1 \tag{5.5}$$

siendo $\delta(Q, w)$ la transición que parte del estado Q con el símbolo w (en este caso, como ya se comentó, cada palabra del vocabulario se considera un símbolo)

y $\Pr(\delta(Q, w))$ la probabilidad asociada a dicha transición.

Volviendo al algoritmo de inferencia de los autómatas, si utilizamos modelos de n-gramas como entrada para éste entonces dicho algoritmo puede simplificarse un poco. Por un lado, podemos eliminar el paso 4a, ya que todos los estados habrán sido creados durante el paso 3 debido al propio proceso de entrenamiento del modelo. Además, el bucle del paso 3c puede obviarse porque como en el formato ARPA los i-gramas vienen en orden creciente de i realmente sólo habría que incluir el estado y la transición que permiten dar cuenta de la conjunción del penúltimo y el último símbolos de la secuencia considerada. Del mismo modo, los pasos 3a y 3b sólo deberán ejecutarse para el caso de los 1-gramas, ya que, si existe un i-grama con $i > 1$ que comience por un cierto símbolo forzosamente el 1-grama correspondiente a dicho símbolo habrá tenido que aparecer previamente. Por último, dado que hemos eliminado el paso 4a, todos los estados del autómata serán finales, ya que, como se expuso anteriormente, $F_k(S) = I_k(S)$, aunque esta idea volverá a ser considerada un poco más adelante.

Por otra parte, en los modelos de n-gramas suelen existir dos palabras especiales que simbolizan el inicio y final de frase, que se conocen con el nombre de *context cues* (suelen representarse respectivamente como <s> y </s>). La función de ambas palabras tiene mucho que ver con su posición dentro de la frase: mientras que el *context cue* inicial sirve para inicializar el contexto de la primera probabilidad condicional (es decir, intentaremos saber la probabilidad de que una cierta palabra pueda comenzar una frase) el final nos informará de la probabilidad de que el último n-grama reconocido pueda concluir una frase. De acuerdo con estos hechos, existen varias implicaciones que tienen cada uno de ellos en la representación del modelo de lenguaje que habrá que tener en cuenta.

Primero veremos qué modificaciones debemos realizar sobre las ideas expuestas considerando únicamente el *context cue* del final de frase. Aunque un poco más arriba se dijo que todos los estados del autómata construido serían finales, si tenemos en cuenta la palabra especial de final de frase esto deja de ser totalmente cierto, ya que justamente el único estado final será el correspondiente a dicha palabra especial. Así, cuando el reconocedor haya terminado de dar cuenta de los *frames* que le hayan llegado por la entrada y quiera terminar el reconocimiento, forzará una transición al estado final, aplicando, eso sí, la probabilidad correspondiente $\Pr(</s>|w_{p-n+1}...w_p)$, siendo p la longitud de la frase reconocida.

Si consideramos ahora el *context cue* inicial, vemos que como éste constituye una inicialización del contexto antes de comenzar el proceso de reconocimiento, lo que deberá hacerse es asignar el estado inicial del autómata al que precisamente identifique a esta palabra. De este modo, la primera probabilidad del modelo de lenguaje que se aplique será la correspondiente a $\Pr(w_1|<s>)$, siendo w_1 la primera palabra que se intente reconocer.

Así, el algoritmo 5.2 es una adaptación del 5.1 teniendo en cuenta los hechos expuestos. Puede verse que el coste temporal del algoritmo 5.2 es lineal con la talla del modelo de n-gramas si consideramos que la operación de unión de conjuntos tiene coste $\mathcal{O}(1)$, ya que sólo se analiza cada i-grama una sola vez y para cada uno de ellos se ejecutan 1 ó 2 operaciones a lo sumo.

Algoritmo 5.2 Algoritmo para la obtención de un AFD A para el menor lenguaje k-TSS que contiene al conjunto de muestras de aprendizaje S, siendo éste un modelo de lenguaje basado en n-gramas con *context cues* inicial y final

---

Nota: En este algoritmo se asume que las secuencias vienen en orden creciente de longitud. Llamaremos <s> al *context cue* inicial y </s> al final.

1. $Q = \{\lambda\}$; $\delta = \varnothing$;
2. **Para cada** 1-grama a_1 **hacer**
   a) $Q = Q \cup \{a_1\}$;
   b) $\delta = \delta \cup \{\{\lambda, a_1, a_1\}\}$;
3. $q_0 =$<s>; $\Sigma(S) =$ conjunto de 1-gramas; $F =$</s>;
4. **Para cada** i-grama $a_1a_2...a_i$ con $2 \leq i < n$ **hacer**
   a) $Q = Q \cup \{a_1...a_i\}$;
   b) $\delta = \delta \cup \{\{a_1...a_{i-1}, a_i, a_1...a_i\}\}$;
5. **Para cada** n-grama $a_1a_2...a_n$ **hacer**
   a) $\delta = \delta \cup \{\{a_1...a_{n-1}, a_n, a_2...a_n\}\}$;
6. **Devolver** $A = \{Q, \Sigma(S), \delta, q_0, F\}$;

Existe un caso que hasta ahora se ha dejado de lado pero que hay que contemplar y resolver: el de los modelos de 1-gramas. En estos modelos únicamente se dispone de la probabilidad de aparición de cada palabra y no podemos utilizar el formalismo de los k-TSS porque sería imposible crear los conjuntos de prefijos y sufijos permitidos. Sin embargo, si consideramos la ecuación 5.5, podemos representar este tipo de modelo mediante un autómata con un solo estado, etiquetado con λ, y una transición partiendo de él por cada palabra del vocabulario, junto con la probabilidad correspondiente [19]. De este modo el autómata tiene la misma estructura que los obtenidos mediante el algoritmo 5.2 y pueden utilizarse las operaciones que se definan sin importar cuál sea la n que defina al modelo de n-gramas.

Por tanto, y considerando todo lo expuesto hasta ahora, un nodo cualquiera del grafo que representa el autómata deberá tener la siguiente información asociada:

- El identificador del estado, que será necesario durante el proceso de construcción del autómata. Antes de iniciar la etapa de reconocimiento puede liberarse la memoria de estos identificadores porque ya no serán necesarios.
- Una lista (ordenada por probabilidad) de transiciones que parten de ese nodo, indicando en cada una de ellas el estado destino de ésta y la probabilidad y el símbolo con el que se efectúa.

Posteriormente se añadirá más información a cada nodo, pero antes deberá considerarse lo expuesto en la siguiente sección.

Para finalizar, y recordando la ecuación 5.4 donde se introdujo el concepto de *Grammar Scale Factor*, podemos ver que todas las probabilidades del modelo vendrán multiplicadas por una cierta constante que habrá sido proporcionada por el usuario. Para aumentar la eficiencia en tiempo de reconocimiento este producto puede hacerse (y así es como se ha implementado en el reconocedor) durante la construcción del autómata, no teniendo así que hacer ninguna multiplicación durante el proceso de reconocimiento.

5.3. Transiciones por *back-off*

En este capítulo, hasta el momento, se ha asumido que en el modelo de lenguaje se encuentran todos los i-gramas posibles que puede decir el locutor, es decir, que a cada uno de ellos se le ha asignado una probabilidad de aparición no nula. Sin embargo en la práctica esto normalmente no es así, ya que en el conjunto de frases que se utilicen para entrenamiento del modelo, por más que se intentará que aparezcan las secuencias más comunes para el ámbito para el que se construye el reconocedor, puede ocurrir que alguna de ellas no aparezca o que incluso el locutor diga una secuencia que ni siquiera se haya contemplado durante la construcción del corpus de entrenamiento. A modo de ejemplo, destacar una reseña histórica que aparece en [9]. Durante la pasada década de los 70 algunos investigadores de IBM llevaron a cabo un experimento para el que dividieron un corpus de descripciones de patentes basado en un vocabulario de 1000 palabras en dos subconjuntos para entrenamiento de un reconocedor y test de éste, con 1500000 y 300000 palabras cada uno respectivamente. Resultó que al analizar los trigramas presentes en cada uno de los conjuntos se encontró que el 23 % de los que aparecían en el test no habían sido vistos ni una sola vez en el conjunto de entrenamiento. Si no pusiéramos remedio a este hecho, esto implicaría que no podríamos evitar que durante la fase de reconocimiento al menos un 23 % de las palabras fueran erróneas.

Una solución para este problema es utilizar lo que se conoce como técnicas de suavizado (o *smoothing* en inglés). Estas técnicas intentan no anular la probabilidad de las secuencias no vistas durante el entrenamiento, pero sí darles

una probabilidad suficientemente baja como para que no interfieran demasiado cuando lo óptimo sería utilizar aquellas que sí aparecen en el modelo de lenguaje.

Una técnica de suavizado sencilla y que ilustra muy bien la idea del párrafo anterior es la conocida como de *descuento absoluto* (o *absolute discounting*), que consiste en restar una pequeña constante a todas las probabilidades mayores que cero que aparezcan en el modelo y repartir equitativamente la masa de probabilidad resultante entre todas aquellas muestras no vistas. Por supuesto, hay métodos mucho más sofisticados que se utilizan en la práctica, pero la idea es justamente la de no dejar que ninguna secuencia tenga probabilidad totalmente nula, permitiendo así que todas las posibles frases formadas con palabras del vocabulario puedan ser reconocidas.

Además de algún método de suavizado, en la representación del modelo de lenguaje se ha utilizado también la técnica que recibe el nombre de *back-off*, y que puede ser definida mediante la siguiente ecuación recursiva (siendo w_1 la palabra más antigua de la historia y w_k la más reciente):

$$\Pr(z|w_1...w_k) = \begin{cases} \Pr(z|w_1...w_k) & si\ \Pr(z|w_1...w_k) > 0 \\ \beta(w_1...w_k) \cdot \Pr(z|w_2...w_k) & en\ otro\ caso \end{cases} \quad (5.6)$$

En esta ecuación el parámetro $\beta(w_1...w_k)$ es un factor obtenido mediante el método de suavizado y que suele ser menor que 1, de forma que actúa como una penalización que se aplica sobre la probabilidad de intentar transitar con dicha palabra "olvidando" la más antigua de la historia. Si en vez de probabilidades consideramos sus logaritmos la ecuación quedaría:

$$\log \Pr(z|w_1...w_k) = \begin{cases} \log \Pr(z|w_1...w_k) & si\ \Pr(z|w_1...w_k) > 0 \\ \log \beta(w_1...w_k) + \log \Pr(z|w_2...w_k) & en\ otro\ caso \end{cases} \quad (5.7)$$

Como ya se expuso anteriormente, la penalización por *back-off* ($\log \beta$) nos la proporciona directamente el fichero ARPA que se da como entrada al reconocedor, así que sólo tendremos que preocuparnos de guardarla correctamente en la estructura de datos que hemos diseñado para el modelo de lenguaje. Como puede verse, esta penalización depende únicamente de la historia que hayamos considerado hasta el momento, la cual precisamente constituye el identificador de cada uno de los nodos del autómata, por lo que dicho valor podrá guardarse junto con el resto de información relativa a los nodos. Además, el hecho de que esta historia es exactamente la misma que aparece en la parte derecha de la probabilidad condicional y que ésta tiene longitud menor que n (el valor que identifica al modelo de n-gramas) explica por qué las secuencias más largas del modelo de lenguaje no tienen valor de *back-off* asociado, tal y como se comentó

cuando se habló del formato de los ficheros ARPA. Por tanto, en la etapa de reconocimiento si queremos saber la probabilidad de transitar desde un estado del modelo de lenguaje (es decir, teniendo en cuenta una cierta historia) con una palabra determinada simplemente tendremos que ver si existe alguna transición desde él con dicha palabra, ya que la existencia de una transición en el autómata implica probabilidad no nula. Si existe, aplicamos la probabilidad de dicha transición y continuamos con el reconocimiento; si no, calculamos el estado equivalente a no considerar la palabra más antigua de la historia y repetimos el proceso. Para mejorar la eficiencia y no tener que calcular varias veces el estado al que se llegaría desde uno determinado aplicando *back-off* junto con la información de cada nodo se ha almacenado también un puntero a dicho estado, convirtiéndose entonces esta transición en una asignación de punteros.

Sin embargo, el caso expuesto de intentar una transición desde un cierto estado con una palabra determinada será un hecho que no se producirá en nuestro reconocedor dada la filosofía *top-down* seguida en su implementación. Aunque se explicará más adelante con más detalle, lo que realmente ocurrirá es que el propio algoritmo de reconocimiento pedirá al modelo de lenguaje que informe de las posibles palabras que puedan seguir a una dada y que superen una probabilidad determinada[4] y éste recorrerá la lista de transiciones del estado que indique la historia para devolver las palabras demandadas, aplicando transiciones por *back-off* cuando sea necesario.

[4]Si no se diera esta probabilidad umbral realmente se devolverían todas las palabras que constituyen el vocabulario del modelo porque se podría ir transitando por *back-off* hasta devolver todos los 1-gramas.

Capítulo 6

El algoritmo de búsqueda

6.1. Representación del espacio de búsqueda

Resumiendo lo expuesto en los tres capítulos anteriores tenemos que, por un lado, los modelos acústico-fonéticos los hemos representado como Modelos Ocultos de Markov, los cuales pueden verse como grafos dirigidos ponderados con pesos asociados también a los nodos (correspondientes a las distribuciones de probabilidad de emisión). Por otro lado tenemos el conjunto de transcripciones fonéticas de cada palabra, las cuales han sido modeladas como varios *tries*, en concreto uno por palabra. Por último, se ha decidido tratar el modelo de lenguaje como un autómata estocástico que representa un lenguaje k-explorable en sentido estricto, con la peculiaridad de que se permiten transiciones por *back-off* para poder dar cuenta de secuencias que no se encontraron durante el aprendizaje de éste.

Si consideramos estos tres elementos en orden inverso al que han aparecido podemos darnos cuenta de que, en lo referente a la topología de las estructuras:

1. Podríamos sustituir las transiciones entre estados del modelo de lenguaje por el árbol de prefijos correspondiente a la palabra que tenga asociada dicha transición, haciendo que los estados finales de dicho *trie* estén conectados por una λ-transición al siguiente nodo del autómata de k-explorables. En el caso de las transiciones por *back-off* la forma de proceder sería similar, pero teniendo en cuenta que el destino de la transición sería el estado al que se llegara con dicha palabra desde el nodo destino del *back-off* y que en algún momento tendrá que aplicarse la penalización correspondiente, de lo cual se hablará más adelante.

2. Cada uno de los nodos de cualquier transcripción fonética podría ser sustituido por el HMM correspondiente al fonema que esté representado en él, concatenándolos apropiadamente según el algoritmo 3.1. De este modo, la unión entre dos palabras se haría concatenando el último estado emisor del HMM del fonema que representa un estado final del trie con el primer estado emisor del primer fonema de la palabra siguiente.

Aplicando estos dos pasos podemos unir los tres tipos de estructuras que hemos expuesto en un solo modelo de Markov, constituyendo éste el objeto sobre el que se efectuará la búsqueda de la frase que maximice la probabilidad de la secuencia de *frames* dada la combinación de modelos acústico-fonético y de lenguaje.

Por un lado, tendríamos las distribuciones de probabilidad de emisión de cada uno de los estados, que no sufrirían ningún cambio con respecto a las determinadas en la fase de entrenamiento.

Considerando ahora lo relativo a las probabilidades asociadas a las transiciones, esta estructura unificada nos brinda una interesante ventaja que es en la que justamente está basada la aproximación de búsqueda *top-down*: puede conocerse en todo momento por el interior de qué palabra se está intentando transitar y, por tanto, aplicar la información proporcionada por el modelo de lenguaje justo cuando se produce el cambio de palabra. En otras palabras, podemos combinar en el mismo paso la probabilidad de salida del último fonema de la palabra que se va a abandonar, la probabilidad de entrada a los primeros estados emisores de la palabra a la que se va a llegar (que en nuestro caso será sólo uno dado el tipo de modelos que utilizamos) y la probabilidad que nos dé el modelo de lenguaje asociada a la transición entre los dos estados correspondientes. Este hecho puede generalizarse para el caso de transiciones por *back-off* simplemente añadiendo a la combinación de probabilidades anterior las penalizaciones oportunas.

Además existen dos casos especiales, correspondientes a los *context cues*. Estas palabras, que simbolizan el inicio y final de frase, no tienen transcripciones fonéticas, por lo que deberá mantenerse su estado del modelo de lenguaje asociado para poder dar cuenta de su existencia (si no fuera así, como el resto de estados se corresponden con nodos de HMMs correspondientes a fonemas, no podríamos representarlas de ninguna manera). De hecho, estos dos estados serán importantes dentro de nuestro modelo de Markov "unificado" ya que constituirán justamente los nodos inicial y final de éste.

Por consiguiente, el modelo de Markov $A = (\Sigma, S, s_0, F, P, Q)$ resultante de todos estos procesos estará compuesto por los siguientes elementos:

- El alfabeto Σ estará compuesto por todos los posibles *frames* que puedan emitirse, por lo que en nuestro caso será igual a $\mathbb{R}^{39}$.

- El conjunto de estados S será el resultante de la composición de los estados de los Modelos Ocultos de Markov de la forma expuesta en los dos pasos anteriores y de acuerdo con el algoritmo 3.1. Adicionalmente se considerarán otros dos estados correspondientes a los *context cues* inicial y final.

- El estado inicial s_0 será el asociado al *context cue* inicial.

- El conjunto de estados finales F tendrá un solo elemento y será el estado que represente al *context cue* final.

- La función de transición entre estados con probabilidades asociadas P se habrá obtenido durante el proceso de concatenación de HMMs. La probabilidad correspondiente a cada una de ellas será la que se haya obtenido en dicho proceso, excepto en los cambios de palabra, donde habrá que aplicar las probabilidades del modelo de lenguaje considerando, en su caso, las penalizaciones por *back-off* oportunas.

- La distribución de probabilidad de emisión de cada estado Q será la misma que se haya estimado para dicho estado durante la etapa de entrenamiento.

Sin embargo, esta estructura tiene el inconveniente de que posee mucha información replicada; por ejemplo, como ya pudo verse en la figura 3.2, el modelo del fonema */a/* tuvo que duplicarse dado que se encuentra dos veces en la secuencia fonética */kasa/*. Además, cada vez que apareciera en el modelo de lenguaje una transición con una cierta palabra ésta se reemplazaría por el árbol de prefijos completo, lo que también produciría réplicas de éstos. La solución por la que se ha optado para poder obtener un coste espacial razonable es la de mantener por separado el modelo de lenguaje, las transcripciones fonéticas y los modelos de Markov correspondientes a fonemas de la forma en la que se ha explicado en los tres anteriores capítulos, y hacer que cada estructura de datos tenga funciones que proporcionen la información necesaria para avanzar en la búsqueda, siguiendo una filosofía similar a la que se expuso en la sección dedicada a la interfaz para modelos acústico-fonéticos. Por tanto, el modelo de Markov sobre el que se efectúa la búsqueda nunca llega a construirse explícitamente, sino que está implícito en la información que puede obtenerse de dichas funciones. En concreto, además de la interfaz establecida para los modelos acústico-fonéticos, se han implementado módulos que cumplen los siguientes requisitos:

- Una función que devuelve todos los estados que pueden alcanzarse desde un estado determinado junto con la palabra con la que se transita y el logaritmo de la probabilidad asociada a la transición de forma que dicho valor sea mayor que un umbral que se pasa como parámetro. Esta función tiene en cuenta posibles transiciones por *back-off* y sus penalizaciones asociadas.

- Un método que informa de todos los posibles siguientes fonemas de uno dado.
- Una función que indica si un fonema es final o no.

Los dos últimos módulos ya fueron explicados cuando se expuso la estructura de datos para la representación de las transcripciones fonéticas para cada palabra, mientras que el primero se tratará con más detalle en secciones posteriores.

Tras esta formalización del espacio de búsqueda como un gran modelo de Markov se expondrá un algoritmo apto para su exploración, que posteriormente se adaptará al caso concreto de la obtención de la frase que emitirá el reconocedor como hipótesis de aquello que ha dicho el locutor utilizando una aproximación dirigida por el modelo de lenguaje. Por último se explicará un heurístico conocido como *beam search*, que permite reducir el número de hipótesis consideradas en cada iteración del algoritmo de búsqueda, y la influencia de este heurístico en el proceso de obtención de las palabras que pueden seguir a una historia dada.

6.2. El algoritmo de Viterbi

En el capítulo 3 se expusieron algunas ecuaciones referentes a Modelos Ocultos de Markov que ahora, dado que hemos modelado nuestro espacio de búsqueda justamente mediante este formalismo, podemos aplicar, aunque interpretando algunas de ellas desde un nuevo punto de vista. Por ejemplo, la ecuación 3.5 nos daba la probabilidad de que una cierta secuencia de *frames* haya sido emitida asumiendo que queríamos reconocer un fonema concreto. Pues bien, si ahora interpretamos que la parte derecha de la probabilidad condicional ya no se corresponde con un solo fonema sino con una frase concreta, aplicándola sobre el "modelo integrado" podríamos obtener la probabilidad de haber emitido la secuencia de *frames* captados por la entrada asumiendo que queremos reconocer dicha frase. Es interesante destacar que restringimos la probabilidad condicional a una frase en concreto y no decimos "cualquier frase", ya que lo que obtendríamos con esto sería la probabilidad de que pueda emitirse la secuencia fonética independientemente de la frase, que no es lo que buscamos. No debemos perder de vista que nuestro objetivo será encontrar la hipótesis de frase que maximice la fórmula 2.3 analizando una a una las muestras acústicas que vayan llegando al reconocedor.

Una forma de estimar la probabilidad combinada de los modelos acústico-fonético y de lenguaje para una cierta secuencia de *frames* teniendo en cuenta la representación del espacio de búsqueda que hemos llevado a cabo es utilizando la conocida como aproximación de Viterbi. Esta aproximación consiste en la asimilación de la probabilidad total por aquella del camino con mayor valor de

ésta, teniendo en cuenta que, como se expuso en el capítulo 3, puede demostrarse que en la mayoría de casos existe un camino dentro de los que componen el sumatorio de la ecuación 3.5 con una probabilidad asociada mucho mayor que el resto. Hay que destacar que aunque lo estadísticamente correcto sería utilizar un método que calculara la probabilidad real (existen algoritmos basados en programación dinámica que así lo hacen) lo más utilizado en la práctica es utilizar esta estimación ya que suele dar una idea bastante aproximada de dicha probabilidad con un menor coste, consecuencia de la sustitución de los sumatorios por maximizaciones. Además, si una vez calculada la probabilidad de Viterbi para una secuencia de *frames* recuperamos el camino (la secuencia de nodos del modelo de Markov) que nos ha llevado a ésta podremos saber cuál es la frase que maximiza esta estimación de la probabilidad, lo que precisamente constituía nuestro objetivo.

La probabilidad de Viterbi ($\widetilde{\Pr}$) de que con una secuencia de longitud t de símbolos pertenecientes al alfabeto del modelo (en nuestro caso, los t primeros frames de la entrada) se llegue a un estado s puede calcularse como sigue:

$$\widetilde{\Pr}(x_1..x_t|s) = \begin{cases} p(s|s_0) \cdot q(x_1|s) & si\ t = 1 \\ \text{máx}_{s' \in S}\ \widetilde{\Pr}(x_1...x_{t-1}|s') \cdot p(s|s') \cdot q(x_t|s) & en\ otro\ caso \end{cases} \quad (6.1)$$

De este modo, la probabilidad total de haber emitido la secuencia acústica de longitud n caracterizada por los *frames* $x_1...x_n$ dado un modelo de Markov M sería:

$$\widetilde{\Pr}(x_1...x_n|M) = \max_{f \in F, s \in S} \widetilde{\Pr}(x_1..x_t|s) \cdot p(f|s) \quad (6.2)$$

Este método de aproximación de probabilidades dado un modelo de Markov se conoce como algoritmo de Viterbi. Además, en nuestro caso, como el modelo de Markov que integra los modelos acústico-fonético y de lenguaje sólo tiene un elemento, en concreto el correspondiente al *context cue* final </s>, podemos simplificar la expresión quedando la ecuación 6.3. Esta expresión, junto con la ecuación 6.1, deberían modificarse para poder trabajar con logaritmos de probabilidades, para lo cual simplemente habría que sustituir los productos por sumas. Esto deberá ser así porque, como ya se ha expuesto en repetidas ocasiones, nuestro algoritmo de reconocimiento no utilizará las probabilidades directamente sino sus logaritmos.

$$\widetilde{\Pr}(x_1...x_n|M) = \max_{s \in S} \widetilde{\Pr}(x_1..x_t|s) \cdot p(</\text{s}>|s) \quad (6.3)$$

Por otro lado, en el reconocedor implementado se ha considerado que sólo se permite transitar al *context cue* final desde el último estado emisor del HMM de un fonema que represente un estado final de la transcripción fonética correspondiente o desde uno cualquiera de los estados de los Modelos Ocultos de Markov correspondientes a silencios (preludios de los árboles de transcripciones fonéticas). Además, en este caso, y como ya se expuso en la sección en la que se trató este tema, la palabra a la que pertenecieran estos silencios no se considerará como completada, ya que ningún fonema "real" de ésta ha podido ser reconocido.

Analizando las ecuaciones anteriores puede verse que el coste temporal del algoritmo de Viterbi es $\mathcal{O}(n \cdot |S|)$, ya que existen tantas llamadas recursivas como elementos tiene la cadena de la parte izquierda de la probabilidad condicional y en cada una de ellas hay una maximización que itera para todos los estados del modelo. Si tratamos este algoritmo desde el punto de vista de la programación dinámica podríamos representar los resultados parciales en una matriz con tantas filas como estados tenga el modelo y tantas columnas como elementos la secuencia $x_1...x_n$, de forma que puede rellenarse avanzando columna a columna. Podemos darnos cuenta entonces de que los resultados de cada columna sólo dependen de la anterior, lo que nos permite sustituir esta matriz por dos vectores de tamaño igual a la cardinalidad del conjunto de estados, rellenándolos de forma alternativa utilizando la información que acaba de calcularse para el otro. De este modo, si no consideramos la recuperación del camino, el coste espacial del algoritmo de Viterbi es $\mathcal{O}(|S|)$.

En consecuencia, los algoritmos 6.1 y 6.2 (que deben ejecutarse secuencialmente en este orden) muestran la adaptación de las ecuaciones anteriores al modelo que integra los modelos acústico-fonético y de lenguaje, teniendo en cuenta también las funciones definidas para la representación implícita de éste. Como puede verse, se utiliza la idea expuesta en la sección anterior de hacer uso de la información del modelo de lenguaje tan pronto como sea posible con el objetivo de guiar un poco más la búsqueda y, posiblemente, reducir el número de hipótesis a considerar. Por tanto, podemos decir que el algoritmo de búsqueda empleado en el reconocedor implementado ha sido una versión del algoritmo de Viterbi dirigida por el modelo de lenguaje.

Sobre este algoritmo destacar que el identificador de cada hipótesis está compuesto por tres elementos, a saber:

- Palabra que se está intentando reconocer (w)
- Identificador del fonema dentro de la palabra del que se está intentando dar cuenta (f)
- Identificador del estado del HMM correspondiente a dicho fonema (q)

Este identificador compuesto por tres campos se debe a que en el modelo de Markov integrado sobre el que se realiza la búsqueda cada estado realmente se corresponde a una composición entre los elementos del modelo de lenguaje, las transcripciones fonéticas y el modelo acústico-fonético, por lo que se necesitará una referencia por cada uno de estos tres componentes para poder identificar unívocamente cada estado, lo cual será necesario para poder realizar las maximizaciones pertinentes. Por otro lado, cada una de estas hipótesis, además de este identificador, dispondrá de más información, la cual será expuesta más adelante, junto con más detalles sobre estos tres elementos.

A la vista del algoritmo 6.1 hay varias cosas que pueden llamar la atención, las cuales se explicarán a continuación. Primero, podemos ver que por motivos de claridad en la exposición se han utilizado las probabilidades "tal cual" en vez de sus logaritmos, pero hay que aclarar que en el reconocedor implementado sí se han utilizado los logaritmos, por lo que habría que sustituír todos los productos de este algoritmo por sumas. Otro aspecto que se debe resaltar es el uso de un factor denominado *Word Insert Penalty* (o *WIP*) en el cálculo de la probabilidad asociada a una hipótesis en la que ha habido un cambio de palabra. El *WIP* suele ser un valor comprendido entre 0 y 1, o, en logaritmo, un número negativo (que estará sumando) cuyo objetivo es favorecer las palabras más largas en contra de secuencias fonéticas cortas similares, las cuales quedarían penalizadas por su aplicación reiterada. Este valor, del mismo modo que ocurría con el *Grammar Scale Factor*, deberá ajustarse manualmente para cada corpus concreto por medio de una serie de experimentos.

También puede verse que, como ya se anunció en la sección anterior, se utilizan las tres funciones que se establecieron (junto con las de la interfaz para modelos acústico-fonéticos) para construir de forma implícita el modelo de Markov sobre el que se ejecuta la búsqueda. Sólo destacar por el momento que en la función de obtención de siguientes palabras dado un estado concreto del modelo de lenguaje falta por considerar un parámetro que determinará cuáles son las palabras que se devuelven, ya que, como se comentó en la sección dedicada a las transiciones por *back-off*, si no consideráramos este umbral podríamos devolver todas las palabras del vocabulario como posibles siguientes.

Una función más sobre la que merece la pena hablar es aquella que determina si durante una cierta iteración del algoritmo ya se ha creado alguna hipótesis con un identificador dado. La utilidad de este método viene justificada por el hecho de que si no se había creado ninguna hipótesis con un identificador concreto la primera que se cree con éste no tendrá que ver si su probabilidad es mayor que otra: simplemente se insertará en la lista de hipótesis para la siguiente iteración. Sin embargo, si no fuera la primera cuando se cree deberá verse cuál de las dos tiene mayor probabilidad (la ya existente o la que acaba de crearse) y quedarnos sólo con la que nos indique la maximización. En la implementación del algoritmo esta función se ha construido mediante un *hashing* basado en los tres elementos del identificador (los cuales se explicarán con más detalle un poco más abajo) y una estrategia de encadenamiento de *open addressing* [3].

Algoritmo 6.1 Algoritmo de Viterbi para reconocimiento automático del habla dirigido por el modelo de lenguaje (algoritmo correspondiente a la ecuación 6.1)

Inicialización:

1. hipotesisActuales = {<s>}; /* *La única hipótesis inicial será la que indique el estado que simboliza la palabra <s>* */
2. hipotesisSiguientes = $\varnothing$;

Procedimiento:
hacer

1. f = obtenerSiguienteFrame ();
2. **Para cada** hipótesis $h \in$ hipotesisActuales, **hacer**

   a) **Para cada** $s \in$ sucesores(h) dentro del HMM del fonema en el que se encontraba, **hacer**
   1) Crear una nueva hipótesis h', identificada por $\{h.w, h.f, s\}$;
   2) $h'.prob = h.prob \cdot p(s|h.q) \cdot q(f|s)$;
   3) **Si** $\exists x \in$ hipotesisSiguientes : $id(x) = id(h')$, **entonces**
      - **Si** $h'.prob > x.prob$ **entonces**
        Eliminar x de hipotesisSiguientes e insertar h' en su lugar;
   4) **Sino**
      - Insertar h' en hipotesisSiguientes;

   b) **Si** $h.q$ se corresponde con el último estado del fonema $h.f$, **entonces**
   1) **para todo** s correspondiente a primeros estados de HMMs de fonemas f' tales que $f' \in$ siguientesFonemas ($h.f$), **hacer**
      - Crear una nueva hipótesis h', identificada por $\{h.w, f', s\}$;
      - $h'.prob = h.prob \cdot p(s|h.q) \cdot q(f|s)$; /* *Asumimos que p(y/x) siendo x e y es el último estado emisor de un modelo e y el primero de otro da la probabilidad asociada a la concatenación de modelos* */
      - **Si** $\exists x \in$ hipotesisSiguientes : $id(x) = id(h')$, **entonces**
        - **Si** $h'.prob > x.prob$ **entonces**
          Eliminar x de hipotesisSiguientes e insertar h' en su lugar;
      - **Sino**
        - Insertar h' en hipotesisSiguientes;

   c) **Si** $h.q$ se corresponde con el último estado de un fonema que finaliza alguna de las transcripciones fonéticas para la palabra $h.w$ o estamos en la primera iteración del algoritmo, **entonces**
   1) listaSiguientes = obtenerSiguientesPalabras ($ML(h)$); /* $ML(h)$ *es el estado del modelo del autómata que representa el modelo de lenguaje al que se ha llegado en el proceso de construcción de la hipótesis h* */
   2) **para todo** $w' \in$ listaSiguientes, **hacer**
      - **para todo** s correspondiente a primeros estados emisores de HMMs asociados a fonemas f' tales que $f' \in$ primerosFonemas (w'), **hacer**
        - Crear una nueva hipótesis h', identificada por $\{w', f', s\}$;
        - $h'.prob = h.prob \cdot \Pr_{ML}(w'|ML(h)) \cdot p(s|h.q) \cdot q(f|s) \cdot$WIP; /* $\Pr_{ML}(x|ML(h))$ *es la probabilidad que nos da el modelo de lenguaje de transitar con la palabra x en el autómata desde el estado al que se había llegado con la hipótesis h* */
        - **Si** $\exists x \in$ hipotesisSiguientes : $id(x) = id(h')$, **entonces**
          - **Si** $h'.prob > x.prob$ **entonces**
            Eliminar x de hipotesisSiguientes e insertar h' en su lugar;
        - **Sino**
          - Insertar h' en hipotesisSiguientes;

3. hipotesisActuales = hipotesisSiguientes;
4. hipotesisSiguientes = $\varnothing$;

mientras queden *frames* por procesar;
ejecutar el algoritmo 6.2;

Algoritmo 6.2 Búsqueda de la frase con mayor probabilidad obtenida con el algoritmo de Viterbi (algoritmo correspondiente a la ecuación 6.3)

mejorHipotesis = NULL;
Para cada hipótesis $h \in$ hipotesisActuales, **hacer**

1. **Si** h representa el último estado del último fonema de una palabra o pertenece a un silencio, **entonces**

   a) Aplicar la probabilidad de transición desde ese estado del modelo de lenguaje al *context cue* final.

   b) **Si** la probabilidad resultante es mejor que la que tenía mejorHipotesis, **entonces**

      1) mejorHipotesis = h;

mejorCamino = recuperarCamino (mejorHipotesis);
Devolver mejorCamino y mejorHipotesis.probabilidad;

Por último, vemos que para el almacenamiento de las hipótesis sólo se emplean dos vectores que simbolizan las actuales y las que se están construyendo para la iteración siguiente. Esto nos provocará un problema en la operación de la recuperación del camino con el que se ha conseguido la probabilidad máxima, el cual nos interesa para poder saber la secuencia de palabras con la que se ha obtenido. Si hubiéramos utilizado la idea de la matriz que se expuso anteriormente este proceso de recuperación podría haber sido muy sencillo si hubiésemos guardado junto al valor de cada una de sus celdas un puntero que indique el elemento de la fila anterior que hemos utilizado para calcular el valor de dicha celda. Por el contrario, si reducimos la memoria disponible para el cálculo a sólo dos vectores el uso de esta técnica se hace imposible, aunque podemos adaptarla ligeramente dado que realmente no nos interesa la secuencia completa de estados del modelo de Markov integrado por los que se ha pasado para obtener la probabilidad máxima, sino sólo las palabras (transcripciones ortográficas) que la han propiciado.

Teniendo esto en cuenta, para poder llevar a cabo la recuperación del camino se ha utilizado un contenedor de palabras que han sido reconocidas en algún momento por alguna de las hipótesis del algoritmo de Viterbi, cada una de ellas con un puntero a su predecesora, de forma que la recuperación del camino con el que se ha llegado a alguna de estas palabras consiste simplemente en un recorrido de punteros hacia atrás. Dicho esto así puede parecer algo extraño, pero lo aclararemos a continuación.

A cada hipótesis del algoritmo de Viterbi se le ha añadido un campo (llamémosle p_ant) correspondiente a un puntero a una palabra dentro de este contenedor, concretamente a la última cuyo reconocimiento consiguió completarse en el camino recorrido hasta la creación de dicha hipótesis. Esto implica que cuando

en una hipótesis se completa una palabra se ejecuta una operación de inserción de un elemento en el contenedor, correspondiente a la palabra que acaba de reconocerse, y asignándosele como predecesora aquella indicada por el puntero p_ant de dicha hipótesis. Por supuesto, tras haber realizado esta inserción este puntero deberá apuntar a la palabra que se acaba de insertar, que es la que hará de predecesora de la siguiente palabra que se consiga reconocer a partir de esa hipótesis. Repitiendo este procedimiento con cada palabra reconocida estamos construyendo "caminos" dentro del contenedor, ya que cada palabra está conectada a su predecesora, y recorriendo estos punteros podemos saber cuáles fueron las palabras cuyo reconocimiento llevó a la hipótesis con que una palabra cualquiera fue insertada. De este modo, cuando termine el algoritmo de Viterbi y tengamos la hipótesis con mayor probabilidad asociada con la que se haya llegado al estado final podemos saber cuál es la frase asociada a esta hipótesis con coste lineal con la longitud de dicha frase, $\mathcal{O}(n)$.

Como consecuencia de todo lo expuesto hasta ahora, a continuación se detallan los campos de la estructura de datos que representa una hipótesis del algoritmo de Viterbi implementado (los tres primeros constituyen el identificador de la hipótesis):

- Un entero correspondiente al identificador de la palabra que se está intentando reconocer.
- Un puntero al árbol de transcripciones fonéticas de dicha palabra que informa de qué fonema está activo para esa hipótesis.
- Un entero que indica el identificador del estado del HMM de dicho fonema en el que se encuentra la hipótesis.
- Un real donde se guarda la puntuación que esa hipótesis ha conseguido hasta el momento.
- Un puntero a la última palabra que algún antecesor de esa hipótesis insertó en el contenedor de palabras reconocidas, para poder así recuperar el camino en caso de que ésta dé lugar a la hipótesis ganadora.
- Un puntero al último estado del autómata que modela el modelo de lenguaje que se ha visitado, para poder conocer las posibles siguientes palabras que se intentarán reconocer y sus logaritmos de probabilidad asociados.
- Un real que indica el valor que tuvo la última transición que se llevó a cabo en el autómata del modelo de lenguaje. El objetivo de guardar este número es para poder restarlo si al final del análisis de todos los *frames* nos encontráramos en un estado perteneciente al preludio de silencios del árbol fonético de la palabra; en dicho caso, como no se habrá reconocido ningún fonema "real" de la palabra, se descartará dicha transición.

- Un puntero al anterior estado del modelo de lenguaje que se visitó antes de la última transición en éste, para poder, en caso de finalizar con una hipótesis correspondiente a un silencio, anular la última transición efectuada e intentar una desde este estado al que representa el *context cue* final.

Para terminar con lo referente al algoritmo de búsqueda, seguidamente se expondrá un heurístico que se ha aplicado al algoritmo de Viterbi para reducir el número de hipótesis a considerar en cada iteración de éste y, en consecuencia, mejorar su eficiencia.

6.3. *Beam search*

El algoritmo de Viterbi consigue encontrar el camino que maximiza la probabilidad de emisión de los *frames* de entrada en el modelo de Markov integrado con coste $\mathcal{O}(n \cdot |S|)$, donde n es el número de vectores acústicos y $|S|$ el número de estados de dicho modelo. Sin embargo, y como puede intuirse, el número de estados del modelo integrado puede llegar a ser muy grande (dependiendo del número de fonemas que se consideren, de la talla del vocabulario y de cuántas transcripciones alternativas tenga cada palabra), lo que provocaría que en cada iteración se analicen muchas hipótesis, redundando así en una baja velocidad de ejecución. Es un hecho que en muchas de las aplicaciones del reconocimiento automático del habla interesa que la emisión de una hipótesis por parte del programa se haga en tiempo real, por lo que, de alguna manera, deberá aumentarse la velocidad de este proceso, y una de las formas de incrementar esta velocidad es utilizando el heurístico conocido como *beam search* o búsqueda en haz.

La idea en la que se basa el *beam search* es en asumir que hipótesis que en una determinada iteración han conseguido una puntuación bastante inferior a aquella que obtenga el máximo no contribuirán al cálculo del camino de máxima probabilidad, por lo que podrán ser ignoradas. En la práctica existen varias maneras de "podar" estas hipótesis; una de ellas es la conocida como *histogram pruning* (o poda por histograma), y consiste en la ordenación de las hipótesis de mayor a menor puntuación, quedándose con las x primeras (siendo x un número determinado a priori) para su procesamiento cuando se obtenga el siguiente *frame*. Otra de estas técnicas, y la que precisamente se ha implementado en el reconocedor, es la que considera como hipótesis válidas para la siguiente iteración aquellas que obtienen una probabilidad mayor o igual a la del máximo multiplicada por un cierto número mayor que 1 (o el máximo más una cierta constante si trabajamos en logaritmos, como es nuestro caso). Cabe destacar que esta constante que se sumará al máximo de la iteración será otro valor que habrá que ajustar empíricamente, al igual que el *Word Insert Penalty* y el *Grammar Scale Factor*. Para el cálculo del conjunto de hipótesis que sobrevivan

a esta poda se ha considerado una aproximación voraz que se explicará con más detalle un poco más abajo.

No obstante, antes de pasar a exponer cómo se ha implementado la búsqueda en haz en el reconocedor es interesante destacar que el uso de este heurístico nos hace asumir el riesgo de poder perder en algún momento la hipótesis que nos lleve al camino de probabilidad máxima, obteniendo por tanto un subóptimo como resultado al final del algoritmo. Esto se debe a que hipótesis que han dejado de considerarse debido a la poda podrían haber conseguido, con el transcurso de los *frames*, recuperar puntuación y "adelantar" a aquellas que la precedían en probabilidad, cabiendo así la posibilidad de que alguna de ellas fuera finalmente la que mejor valor cosechase. Sin embargo, a pesar de esta posible pérdida de la secuencia de palabras óptima, en la práctica el *beam search* es una técnica muy utilizada dados los buenos resultados que obtiene y la mejora de eficiencia que produce.

Como se ha comentado, para la implementación del *beam search* en el reconocedor se ha utilizado una aproximación voraz. En esta técnica se guarda en una variable auxiliar el valor asociado a la hipótesis de mayor probabilidad que se ha considerado hasta el momento en la presente iteración, permitiendo que sólo puedan avanzar aquellas hipótesis con puntuación mayor que este máximo más un margen determinado, el cual será un número negativo ya que estamos hablando de logaritmos de valores menores que 1. Sin embargo, podría ocurrir que el máximo absoluto de la iteración se consiguiera ya bastante avanzada ésta, por lo que habríamos pasado por una serie de "máximos relativos" que, efectivamente, habrán hecho su función no dejando pasar aquellas hipótesis que no sean lo suficientemente prometedoras con respecto de ellos, pero que puede resultar que incluso estos máximos locales no cumplan la condición de tener una probabilidad asociada mayor que el máximo absoluto de la iteración más el margen de holgura. En otras palabras, puede ocurrir que se hayan "colado" hipótesis en la lista que se va a considerar en la iteración siguiente que, si hubiera sido la correspondiente al máximo absoluto la primera que se hubiera calculado, no habrían llegado nunca a entrar en la lista. Para solucionar este problema, dado que al finalizar una iteración sí podemos saber cuál ha sido la puntuación máxima alcanzada por una hipótesis durante el transcurso de ésta se ha efectuado una segunda poda atendiendo a los siguientes dos criterios:

1. Si la lista de hipótesis a procesar en la iteración actual tiene un número de elementos menor o igual a 150, se considera la totalidad de éstos para llevar a cabo el cálculo de las hipótesis para la iteración siguiente.

2. En caso contrario, se pone como condición para que una hipótesis de la lista de actuales sea procesada que su puntuación sea mayor que la que se detectó para el máximo de esta lista más la constante considerada. La consecuencia de esto es que se obtiene el mismo resultado que si, después de haber calculado todas las hipótesis sin haber podado nos quedáramos

con aquellas que superan el umbral, pero sin el coste de calcularlas todas gracias a haber efectuado la primera poda.

Por otro lado, para la implementación de este método de poda podría haberse inicializado al principio de cada iteración la variable en la que almacenaremos el valor de la puntuación máxima conseguida en el proceso de generación de las hipótesis para la siguiente iteración al valor $-\infty$ (o un número negativo muy grande), lo que aseguraría que al final del proceso de maximización esta variable tomaría siempre el valor de dicho máximo. Sin embargo, para favorecer todavía más el primer proceso de poda, esta variable tomará al inicio de cada una de las iteraciones un valor igual al máximo que se detectó en la iteración anterior más otra constante negativa que también habrá que ajustar empíricamente. Obviamente, al principio de la primera iteración sí tomará un valor asimilable a $-\infty$ ya que no se dispone de ninguna referencia para poder efectuar otra inicialización.

Sin embargo, esta forma de efectuar el *beam search* no tiene sólo influencia en el proceso de poda de hipótesis una vez éstas ya han sido generadas, sino también durante su propia generación. De esto será sobre lo que hablaremos a continuación.

6.3.1. Influencia de los parámetros del *beam search* en el cálculo de las palabras siguientes a un estado determinado del modelo de lenguaje

En la sección 5.3, cuando se habló de las transiciones por back-off, se comentó que, gracias a ellas, desde un estado cualquiera se podría transitar con cualquier palabra a otro estado del autómata que representa el modelo de lenguaje. Esto implicaría que en un modelo con un vocabulario de, por ejemplo, 10000 palabras, cada vez que se llevara a cabo un cambio de palabra se generarían tantas hipótesis como la media de estados iniciales de los *tries* multiplicado por 10000. Para reducir este número se han intentado aprovechar las ventajas que nos brinda el *beam search* combinado con la búsqueda dirigida por el modelo de lenguaje del modo que a continuación se detalla.

Supongamos que h es una hipótesis del algoritmo de Viterbi en la que se va a llevar a cabo un cambio de palabra. Entonces se invocará a la función encargada de devolver una lista con las posibles transiciones desde el estado en el que se encuentre h, informando ésta también de qué nuevos estados se alcanzarían mediante esas transiciones y con qué puntuación (aplicando las penalizaciones por back-off oportunas). Sabemos también que h tendrá también una probabilidad asociada y que en una variable (llamémosla max) tendremos guardado el valor que debe superar una nueva hipótesis para que pueda ser considerada. Por tanto,

se deduce que para que una hipótesis correspondiente a una palabra nueva vaya a ser tenida en cuenta la probabilidad asociada a la transición correspondiente deberá ser de al menos $max - h.prob$. En consecuencia, para mejorar la eficiencia de la función que informa de las transiciones simplemente tenemos que añadirle un parámetro adicional, correspondiente precisamente a este último valor, para hacer que devuelva sólo aquellos cambios de palabra cuya probabilidad asociada sea mayor que él. Además, como se expuso cuando se habló sobre el modelo de lenguaje, la lista de transiciones desde cada estado estará ordenada de mayor a menor probabilidad asociada, lo que nos permitirá no tener que analizar todas las transiciones que parten desde un determinado estado si encontramos que una de ellas ya no supera el umbral. Esto es lo que se plasma en el algoritmo 6.3, que muestra cómo se ha implementado esta función. Es importante destacar que se asume que se trabaja con los logaritmos de las probabilidades.

La idea en la que se basa dicho algoritmo es la de recorrer la lista de transiciones que parten de cada estado hasta que encontremos una que no supere el umbral establecido o se llegue al final de la lista. En cualquiera de estos dos casos se intentará una transición por *back-off* y se repetirá el proceso. Además, en el caso de que la misma palabra aparezca en más de una lista de transiciones nos quedaremos con la primera que se haya encontrado (es decir, la de la lista más próxima al estado desde el que partió la búsqueda), de acuerdo con lo que sugiere la fórmula 5.6. Si se detecta que el estado actual es el correspondiente a la raíz del autómata (estado λ) se terminará la búsqueda, así como si la penalización por *back-off* acumulada supera el umbral (lo que implicaría que ninguna de las transiciones que partieran de ese estado u otros alcanzables por *back-off* tampoco lo alcanzaría) o se detecta que ya se han insertado en la lista de siguientes palabras todas las que componen el vocabulario. La consecuencia de todo esto es que podemos acotar el número de transiciones que se disparan desde un determinado estado analizando en total sólo unas pocas más de las que constituyen este conjunto.

Por otro lado, destacar también un aspecto negativo de este algoritmo, y es que en el caso peor podría llegar a tener un coste $\mathcal{O}(n \cdot |V|)$, donde n es el valor que determina los n-gramas del modelo de lenguaje y $|V|$ es la talla del vocabulario. Este caso, extremadamente raro, podría darse cuando una hipótesis se encuentre en un estado del modelo de lenguaje que tenga asociada una historia de longitud $n-1$ y todas las transiciones que partan de él y de los estados a los que se llegue por *back-off* tengan exactamente $|V|-1$ transiciones salientes, las mismas en todos los casos excepto en el estado que simboliza la raíz, donde por la propia construcción del autómata todas las palabras son accesibles. Además, el umbral que se le pasa al algoritmo es tan amplio que permite analizar todas las transiciones y todos los estados a los que se llega por *back-off*. Este escenario negativo nos haría analizar casi todas las palabras del vocabulario una y otra vez, hasta llegar a la raíz, donde por fin conseguiríamos la hipótesis que nos falta. Sin embargo, y como ya se ha dicho, este caso es muy raro, y no se prevé que ocurra en la práctica. De hecho, lo que se estima es que el número de transiciones

Algoritmo 6.3 Método para calcular las palabras siguientes a un estado dado teniendo en cuenta la información proporcionada por el *beam search*, asumiendo que la lista de transiciones desde un estado está ordenada de mayor a menor probabilidad asociada

Entrada: e = estado del modelo de lenguaje en el que se encuentra la hipótesis;
umbral = $max - h.prob$;

1. listaSiguientes = $\varnothing$; numSiguientes = 0; penalizacion = 0; e' = e; insertadas = $\varnothing$;
2. **Mientras** penalizacion $\geq$ umbral **&&** numSiguientes $<$ tamVocabulario **hacer**
   - *a*) **Para cada** transición directa t desde el estado e', **hacer**
     1) **Si** t.prob+penalizacion $\geq$ umbral, **entonces**
        - **Si** t.palabra $\notin$ insertadas, **entonces**
          - Añadir t.palabra a insertadas;
          - Añadir la tupla {t.palabra, t.prob+penalizacion} a listaSiguientes;
          - numSiguientes++;
     2) **Sino**
        - **break**;
   - *b*) penalizacion = penalizacion + e'.penalizacionBackOff;
   - *c*) **Si** e' = raíz del autómata de k-explorables (estado λ), **entonces**
     1) **break**;
   - *d*) **Sino**
     1) e' = e'.destinoBackOff;
3. **Devolver** listaSiguientes;

que se analizarán en el transcurso del algoritmo será ligeramente mayor que el número de transiciones que efectivamente se devuelvan al final de éste.

Señalar también que el coste espacial de este método es lineal con el número de palabras del vocabulario por dos motivos. Por una parte, la lista de palabras siguientes que se devuelve como resultado del algoritmo puede llegar a tener talla igual al número de palabras del vocabulario, si las devolviéramos todas como posibles siguientes. Por otro lado, con el objetivo de poder ejecutar la operación de discriminación de pertenencia en tiempo $\mathcal{O}(1)$, se dispone de un vector de tamaño $|V|$, en el que las palabras correspondientes a transiciones que ya se han insertado en la lista que se devolverá como resultado se marcan como

"visitadas". De este modo, utilizando como índice en el vector el identificador de cada palabra, el cual es un número entero, puede conocerse esta pertenencia en tiempo constante.

Capítulo 7

Resultados experimentales

Con el objeto de comprobar el correcto funcionamiento del reconocedor y el cumplimiento de los requisitos especificados en la sección 1.2 se ha llevado a cabo una batería de experimentos con el corpus Albayzin [12, 13] para intentar determinar los valores óptimos de los parámetros del reconocedor de forma que se obtenga el mínimo error para el subconjunto de test de dicho corpus. A continuación se detallarán dichos experimentos y se expondrán los resultados obtenidos.

Sin embargo, antes de llevar a cabo los experimentos es interesante definir qué entendemos por error en el ámbito del reconocimiento automático del habla. En esta disciplina pueden definirse distintos tipos de errores, como el porcentaje de frases incorrectas o de palabras mal transcritas. El primero de ellos no suele utilizarse ya que, como se expuso hace algunos capítulos, normalmente no trataremos de que nuestra transcripción sea perfecta, sino de cometer un número de fallos razonable. Tampoco es normal usar el segundo de los dos citados, ya que podría darse el caso de que la transcripción no tuviera el mismo número de elementos que la frase original. Esto nos lleva a pensar en utilizar técnicas de programación dinámica, como la distancia de edición con respecto a la trascripción correcta, para poder determinar el error cometido. Basado justamente en esto tenemos la medida de error conocida como *Word Error Rate* (o *WER*), la cual se obtiene a partir de la expresión 7.1, donde S, B e I representan respectivamente el número de sustituciones, borrados e inserciones obtenidos a partir de la distancia de edición y L es la longitud de la frase de referencia (transcripción correcta). Además, si extendemos esta medida de error para un conjunto de frases, los tres primeros valores se referirán al número total de operaciones de ese tipo realizadas en todo el conjunto y L será la longitud total de las frases de referencia. Esta será la medida de error que utilizaremos en los posteriores experimentos.

$$WER = \frac{S + B + I}{L} \tag{7.1}$$

Destacar también que estos experimentos se han llevado a cabo en un ordenador de uso común con un procesador a 2.4 GHz y 1 GB de RAM, bajo el sistema operativo Ubuntu 8.04.

7.1. Ajuste de los parámetros del reconocedor

Como se ha expuesto a lo largo de la memoria, el reconocedor implementado tiene cuatro parámetros que deben ajustarse empíricamente los cuales son:

- El *Grammar Scale Factor* (al que llamaremos $alfa$), el cual nos brinda la posibilidad de darle más importancia a las probabilidades del modelo de lenguaje o a las del modelo acústico-fonético, según el valor que se le asigne.
- El *Word Insert Penalty* (al que llamaremos *wip*), que constituye un "castigo" para las hipótesis que intentan efectuar demasiados cambios de palabra.
- La constante negativa que se sumará al máximo encontrado hasta un determinado momento de una iteración del algoritmo de Viterbi para poder aplicar el *beam search*. A este valor lo llamaremos *beam*.
- La constante negativa que se sumará a la puntuación máxima de una hipótesis en la iteración anterior del algoritmo para inicializar el máximo de la iteración actual. A este valor lo llamaremos *ci*.

Para intentar determinar los valores de estos cuatro parámetros que consiguen el mínimo error en reconocimiento utilizando como entrada el subconjunto de test de la base de datos Albayzin se ha seguido la siguiente estrategia. Primero, se han fijado tres de los cuatro parámetros a valores que se han considerado razonables *a priori*, quedando de momento el restante sin determinar. Con el objetivo de asignar un valor al parámetro que todavía no lo tiene se ha efectuado un barrido para posibles asignaciones a éste, dejando el resto sin modificar durante este proceso e intentando encontrar así el mínimo de error que podría obtenerse de este modo. Tras encontrar este mínimo, se elige otro valor del conjunto y se repite el proceso, iterando de este modo hasta encontrar una tasa de error razonable, considerando que para el tamaño del vocabulario la base de datos utilizada (2898 palabras) el funcionamiento debería ser en aproximadamente tiempo real, por lo que podría ocurrir que tengamos que conformarnos con subóptimos de la función de error condicionados por el tiempo de cómputo empleado para reconocer.

Para llevar a cabo el proceso descrito en el párrafo anterior se decidió emplear un modelo de lenguaje basado en trigramas entrenado con el subconjunto de entrenamiento de Albayzin y unos modelos acústico-fonéticos también entrenados con dicho corpus. Además, dado el elevado número de pruebas que podría llevarnos todo este proceso de asignación de valores a los parámetros, se restringió el conjunto de test a un subconjunto de 100 elementos tomados de las frases originales que incluye la base de datos para efectuar pruebas. Por último, decir que como muestra para estimar la velocidad del reconocedor para una cierta parametrización se utilizó el correspondiente a la primera frase del conjunto de test original, la cual se transcribe como *"Francia, Suiza y Hungría ya hicieron causa común"* y tiene una duración de 3.99 segundos.

Los valores iniciales de los parámetros han sido los siguientes:

- $alfa = 5$
- wip = -15
- ci = -350
- $beam$ = parámetro cuyo barrido efectuaremos en primer lugar, utilizando el conjunto de valores {-25, -50, -75, -100}

Y el resultado del primer barrido:

beam	-25	-50	-75	-100
WER (%)	54.25	20.47	15.44	14.94
tiempo (s)	<1	≈ 1	2.53	≈ 4

Tabla 7.1: Resultados para el primer barrido de valores del parámetro *beam*

Puede verse que alrededor del valor -75 se obtiene un reconocimiento en menos de tiempo real, mientras que para -100 ya se tarda un equivalente a la duración de la frase. Por tanto, se decidió explorar el tramo de posibles valores entre ambos números, para ver qué resultados se obtienen, los cuales se muestran en la tabla 7.2.

beam	-75	-80	-85	-90
WER (%)	15.44	15.44	14.65	14.65
tiempo (s)	2.53	3.56	3.28	3.79

Tabla 7.2: Resultados para el segundo barrido de valores del parámetro *beam*

Como se aprecia en esta tabla, a medida que se disminuye el valor de *beam* (se hace más negativo) se obtiene menor error de reconocimiento, aunque aumenta

el tiempo necesario para éste. Esto es debido a que cuanto menor sea este valor menos hipótesis se dejan fuera del procesamiento por culpa del *beam search* y, por tanto, menos probabilidades hay de perder el óptimo durante dicha poda. Del mismo modo, como en cada iteración del algoritmo se consideran más hipótesis el tiempo de reconocimiento va aumentando.

Tras efectuar estos experimentos, se decidió fijar de momento el valor de *beam* a -85, ya que proporciona una tasa de error suficientemente baja comparada con las demás con un coste temporal todavía bastante por debajo de tiempo real. Ahora analizaremos la influencia del factor *ci* realizando pruebas con los valores {-200, -250, -350, -450}, dejando el resto de parámetros fijos. A continuación se muestran los resultados obtenidos.

ci	-200	-250	-350	-450
WER (%)	37.24	14.65	14.65	14.65
tiempo (s)	3.22	4.45	3.28	3.46

Tabla 7.3: Resultados obtenidos durante la variación de valores del parámetro *ci*

Lo que la tabla anterior nos indica es que si le damos al parámetro *ci* un valor demasiado bajo en algunos casos estaríamos inicializando el máximo de la siguiente iteración del algoritmo a un valor demasiado alto, perdiendo así la solución óptima o incluso tal vez todas las hipótesis, si ninguna de ellas superara el umbral establecido al inicio de la iteración. Sin embargo, vemos que llegado el valor -250 el porcentaje de error se estabiliza, lo que quiere decir que este factor ha dejado de influir en la búsqueda de la solución óptima. En consecuencia, a la vista de lo expuesto en la tabla, se ha decidido fijar el parámetro *ci* a -350, que ha sido justamente el valor que menor tiempo ha proporcionado para el proceso de búsqueda.

El siguiente paso en la experimentación fue variar el parámetro correspondiente al *Word Insert Penalty* (*wip*) para analizar su influencia en el reconocimiento y ajustarlo para intentar obtener el menor error posible. En este caso, manteniendo constantes los otros tres parámetros se han hecho experimentos para valores de *wip* iguales a {-5, -7.5, -10, -15}. El motivo de esta selección de valores aparentemente tan caprichosa es que empezó a disminuirse el valor de *wip* comenzando en -5, pero como se vio que la tasa de error empeoraba a medida que se avanzaba se decidió, para agilizar los experimentos, no hacer más pruebas. En la tabla 7.4 se muestran los resultados obtenidos para estos experimentos.

wip	-5	-7.5	-10	-15
WER (%)	12.86	12.86	13.53	14.65
tiempo (s)	5.76	4.31	3.61	3.28

Tabla 7.4: Resultados obtenidos durante la variación del parámetro *wip*

Lo que muestra la tabla anterior es que cuanto mayor es el valor de *wip* (menor en valor absoluto) obtenemos una tasa de error más baja, a costa de un mayor tiempo de ejecución, hasta alcanzar el valor de -7.5, a partir del cual parece que el *WER* no mejora más. En consecuencia, hemos decidido tomar como bueno el valor -7.5, el cual nos proporciona una tasa de error sólo un poco superior con un tiempo bastante más bajo que con el otro valor. El siguiente valor que se probó, correspondiente a wip = -10, ya hace aumentar demasiado el error, así que por ahora no se ha considerado adecuado. Por tanto, como se ha dicho, en los experimentos que vienen a continuación se utilizará un valor de -7.5 para el parámetro *wip*, pero teniendo en cuenta que si no conseguimos bajar el tiempo de reconocimiento de las frases ajustando el resto de valores (además del error de reconocimiento) deberíamos plantearnos asignar a esta variable el valor -10.

Tras efectuar los experimentos anteriores, nos centramos en determinar el valor del parámetro que nos falta, correspondiente al *Grammar Scale Factor* ($alfa$). Para ello, se dieron valores en el rango $[4, 17]$, obteniendo los resultados que a continuación se muestran:

$alfa$	4	5	6	7	8	9	10	11	12	13	14	15	16	17
WER (%)	16.77	12.86	10.96	9.39	7.83	6.38	6.15	4.70	3.80	3.36	2.57	2.35	2.80	3.57
tiempo (s)	5.73	4.31	3.45	3.90	3.25	2.45	2.18	2.69	1.71	1.70	1.89	1.20	1.07	1.29

Tabla 7.5: Resultados obtenidos modificando el parámetro $alfa$, dejando el resto estáticos

En la tabla 7.5 puede verse que a medida que aumentamos el valor del parámetro $alfa$ conseguimos disminuir tanto el *WER* como el tiempo de reconocimiento de la frase que estamos utilizando como muestra. Sin embargo, este hecho puede estar directamente relacionado con la propia naturaleza de la base de datos que estamos utilizando para hacer las pruebas.

Nuestros experimentos se han realizado utilizando el corpus fonético de la base de datos *Albayzin*, que suele emplearse para probar sistemas de segmentación fonética. Aunque también se puede utilizar para probar sistemas de reconocimiento automático del habla, debemos comentar que, mientras que sí es cierto que la intersección entre los locutores de los subcorpus de entrenamiento y de test está vacía, sólo existen unas pocas frases (algo más de una decena) que se encuentran únicamente en el conjunto de test y no en el de entrenamiento. Para que el corpus fuera realmente apto para evaluar el error cometido por el reconocedor implementado este número debería ser mucho mayor, con el objetivo de que el proceso de reconocimiento no estuviera demasiado "guiado" por el modelo de lenguaje y, en consecuencia, obtuviéramos una tasa de error mucho más baja de la que realmente se obtendría si un usuario humano utilizara el sistema para, por ejemplo, funciones de transcripción automática de textos hablados.

Considerando lo expuesto en el párrafo anterior, lo que puede estar ocurriendo a la vista de los resultados de la tabla 7.5 es que, dado que a medida que se aumenta el valor del factor $alfa$ se le está dando más "peso" al modelo de lenguaje con relación al modelo acústico-fonético, es más sencillo reconocer las frases que se encuentran simultáneamente en ambos subconjuntos, ya que la búsqueda está más guiada por los datos del modelo de lenguaje, lo que no sólo propicia un menor error en reconocimiento, sino también un tiempo menor para la ejecución de este proceso debido a la mayor cantidad de podas que facilita el modelo de lenguaje. También ocurre que, como puede deducirse de los últimos valores de dicha tabla, si cargamos demasiado el peso de la búsqueda sobre el modelo de lenguaje, al final se acaba teniendo muy poco en cuenta la parte acústica y, en consecuencia, cometiendo un mayor error, de lo que se desprende que lo realmente interesante es encontrar un equilibrio entre las ponderaciones del modelo acústico-fonético y el de lenguaje, sin importar el hecho de que muchas de las transcripciones ortográficas presentes en el conjunto de test se hayan visto anteriormente durante el proceso de entrenamiento.

De todos modos, destacar también que el objetivo real de estos experimentos es la comprobación del correcto funcionamiento del reconocedor bajo un entorno restringido, así como la constatación de los diversos efectos que pueden tener cada uno de los parámetros de entrada al reconocedor, quedando como trabajo futuro una óptima asignación de valores a éstos en contextos más complejos.

Volviendo a la búsqueda del mejor valor de $alfa$ podemos apreciar que en $alfa$ =15 obtenemos el mínimo error con el menor tiempo de ejecución. Sin embargo, a la vista de la evolución de la tasa de error podemos sospechar que el mínimo real se alcanza entre los valores 15 y 16, por lo que se decidió realizar más experimentos para números en este rango. Los resultados de estas pruebas se muestran a continuación.

$alfa$	15.00	15.25	15.50	15.75	16.00
WER (%)	2.35	2.24	2.24	2.68	2.80
tiempo (s)	1.20	1.18	1.21	1.13	1.07

Tabla 7.6: Resultados obtenidos variando el valor de $alfa$ entre 15 y 16

Tras realizar estos experimentos, se decidió que el mejor valor para el parámetro referente al *Grammar Scale Factor* sería de 15.25, ya que nos proporciona el mínimo *WER*, al igual que 15.50, pero con un menor tiempo de ejecución que éste.

En este momento ya hemos ajustado los cuatro parámetros de los que se habló al principio del capítulo. Sin embargo, ha ocurrido que a veces nos hemos tenido que quedar con algunos valores que, aunque el error que proporcionaban no era mínimo, si seguíamos aumentándolos o disminuyéndolos para seguir

minimizando dicho error el tiempo de ejecución que obteníamos era excesivo. Dado que con la parametrización actual tenemos un tiempo de reconocimiento *off-line* de 1.18 segundos para una frase cuya duración real es de 3.99, podemos volver a considerar estas variables, concretamente *beam* y *wip* para intentar seguir disminuyendo el error.

En primer lugar, se ha vuelto a iterar sobre el parámetro *wip*, cuyo valor hasta ahora era de -7.5, dejando el resto con la configuración que mejor resultado ha dado hasta el momento ($beam$ = -85, ci = -350, $alfa$ = 15.25). Se han probado diversos valores entre -5 y -10, cuyos resultados se exponen en la siguiente tabla:

wip	-5.0	-6.5	-7.5	-8.5	-10.0
WER (%)	2.35	2.35	2.24	2.24	2.57
tiempo (s)	1.22	1.19	1.18	1.16	1.20

Tabla 7.7: Resultados correspondientes a la variacion de *wip* entre -5 y -10

A la vista de los resultados de la tabla, podemos afirmar que no hay mejoras significativas ni por parte del error ni por la del tiempo de reconocimiento, por lo que se decidió mantener el parámetro *wip* con el valor -7.5.

Como se dijo, también se volvió a iterar para nuevos valores de *beam*, obteniendo los resultados de la tabla 7.8. Puede apreciarse en ella que el valor -125 resulta excesivo, ya que aumenta demasiado el tiempo de ejecución debido a que analiza demasiadas hipótesis, así que el nuevo valor de este parámetro será -100.

beam	-85	-100	-125
WER (%)	2.24	2.15	1.90
tiempo (s)	1.18	1.96	5.40

Tabla 7.8: Resultados de los experimentos sobre una nueva variación del parámetro *beam*

En consecuencia, la asignación de valores a los parámetros del reconocedor que utilizaremos en posteriores experimentos será la siguiente:

- $beam$ = -100
- ci = -350
- wip = -7.5
- $alfa$ = 15.25

Cabe destacar que, por más que esta es la mejor combinación de parámetros que se ha podido hallar de forma que existiera un compromiso entre la tasa de error obtenida y la velocidad de ejecución del algoritmo, puede haber estado bastante influida por la inicialización que se eligió. Los valores que se tomaron inicialmente fueron aquellos que se consideraron "razonables", pero eso no les quita una cierta carga de aleatoriedad. Del mismo modo, el uso de un modelo de lenguaje basado en trigramas, así como el empleo de un subconjunto del subcorpus de test en vez de éste en su totalidad pueden haber influido en estos resultados. Es posible que si las decisiones tomadas en estos aspectos hubieran sido diferentes, los valores obtenidos para los parámetros pudieran haber sido distintos, obteniendo en este caso una tasa de error que posiblemente fuera diferente a la que se tiene actualmente y con un tiempo de ejecución que también podría diferir.

7.2. Análisis de la influencia del modelo de lenguaje en el reconocimiento

Una vez hemos encontrado una asignación de valores a los parámetros del reconocedor que nos aporta un error de reconocimiento bajo con un tiempo de ejecución razonable (en este caso, aproximadamente la mitad de lo que dura la frase de entrada), intentaremos ver la influencia del modelo de lenguaje en la etapa de reconocimiento. Para ello utilizaremos esta parametrización y cogeremos como conjunto de test todas las frases del subcorpus de test de la base de datos *Albayzin*, y no sólo 100 como en los experimentos anteriores.

Los experimentos realizados que nos permitirán analizar la influencia del modelo de lenguaje en el reconocimiento son los siguientes. Se considerarán 6 modelos de lenguaje diferentes como entrada al reconocedor, a saber, los entrenados con el subcorpus de entrenamiento de *Albayzin* utilizando desde 1-gramas hasta 6-gramas, y se medirá el *WER* obtenido en reconocimiento y el tiempo de ejecución de la misma frase que se consideró en los experimentos anteriores. De este modo, podremos ver la influencia de la cantidad de información que proporcione el modelo de lenguaje tanto en el error como en el tiempo de ejecución del algoritmo de reconocimiento.

Teniendo en cuenta lo expuesto en la sección anterior sobre la naturaleza del corpus *Albayzin* lo lógico sería pensar que como la mayor parte de transcripciones ortográficas presentes en el conjunto de test también lo están en el de entrenamiento, cuanta mayor información tenga el modelo de lenguaje (mayor sea la n de los n-gramas) menor tasa de error se obtendrá, y además con un tiempo de ejecución menor. En la siguiente tabla se encuentran los resultados de los experimentos realizados para comprobar esta hipótesis.

Modelo de lenguaje	1-gramas	2-gramas	3-gramas	4-gramas	5-gramas	6-gramas
WER (%)	41.64	10.62	1.69	0.98	0.81	0.78
tiempo (s)	6.13	2.70	2.14	1.83	2.11	2.22

Tabla 7.9: Evolución del *WER* y el tiempo de reconocimiento a medida que se aumenta la información contenida en el modelo de lenguaje

Los números presentes en la tabla 7.9 confirman una parte de nuestra hipótesis inicial, concretamente la relativa al *WER*, pero nos hacen ver que en lo referente al tiempo de ejecución estábamos ligeramente equivocados. Por un lado, como podíamos suponer, la presencia de una mayor cantidad de información en el modelo de lenguaje nos ha permitido disminuir la tasa de error hasta menos de un 1 %. Sin embargo, esto no es siempre cierto en la práctica: normalmente a partir de un cierto umbral la tasa de error vuelve a subir, por lo que en la práctica suelen utilizarse modelos basados en bigramas o trigramas, los cuales suelen dar bastante buen resultado.

Por otra parte, vemos que el tiempo de ejecución se ha mantenido en descenso hasta los 4-gramas, volviendo a subir después, en contra de la hipótesis que habíamos formulado en un principio. Esto puede ser debido al algoritmo de devolución de las palabras siguientes a un determinado estado: si consideramos un estado con una historia bastante larga puede ser posible que para devolver todas las siguientes palabras con probabilidad mayor que un cierto umbral tengamos que transitar varias veces por *back-off*, buscándolas en los estados correspondientes. Además, como se comentó en el capítulo anterior, el coste de este algoritmo dependía de la longitud de las secuencias más largas consideradas en el modelo de lenguaje, así que puede ser por esto que el tiempo de ejecución vuelva a subir una vez alcanzado un cierto valor.

Queda como trabajo futuro la prueba del reconocedor con otro corpus donde no haya un porcentaje tan alto de transcripciones ortográficas presentes tanto en el subcorpus de entrenamiento como en el de test para poder contrastar los resultados aquí obtenidos. De todos modos, podemos afirmar que con los distintos experimentos se ha cumplido el objetivo de comprobar el correcto funcionamiento del reconocedor utilizando un corpus concreto.

Capítulo 8

Conclusiones

Como ya se expuso en la introducción, en el presente proyecto se ha desarrollado un software para el reconocimiento automático del habla basado en una filosofía *top-down* e implementado en lenguaje C. La configuración de este programa se efectúa por medio de tres ficheros en los que están contenidas las tres fuentes de conocimiento que pueden verse en la figura 2.1 y las asignaciones de valores a los parámetros del reconocedor que deben ajustarse empíricamente. La salida proporcionada por el reconocedor consta de las transcripciones ortográficas que considera el programa que mejor se corresponden con la señal vocal de entrada teniendo en cuenta los modelos proporcionados.

En los anteriores capítulos se han mostrado los fundamentos teóricos de las decisiones tomadas para el desarrollo de este programa y cómo éstas se han llevado a cabo en la implementación, sin entrar exhaustivamente en detalles del código. Además, se ha comprobado el correcto funcionamiento del reconocedor realizando experimentos con la base de datos *Albayzin*, y analizando la influencia de la cantidad de información proporcionada por el modelo de lenguaje en el proceso de reconocimiento.

Sin embargo, todo esto no quiere decir que el desarrollo del reconocedor esté completamente terminado. Por un lado, uno de los requisitos del sistema era que pudiera reconocer frases utilizando modelos de lenguaje de alrededor de 50000 palabras en menos de 10 veces su duración, requisito que no ha podido probarse debido a la imposibilidad de conseguir un corpus de estas características. Por otro lado, se ha comprobado que los requisitos de funcionamiento se han alcanzado utilizando el corpus *Albayzin*, el cual, como ya se dijo, posee ciertas características que nos resultan favorables. Faltaría ver qué tasa de error y qué tiempo de ejecución se obtienen utilizando otra base de datos de dimensiones similares pero con otras características, lo cual, si no se consiguen los resultados deseados, podría implicar modificaciones en la implementación del programa.

También a lo largo de esta memoria se han ido proponiendo mejoras que podrían hacerse al software desarrollado para, posiblemente, aumentar su rendimiento. Sería interesante desarrollarlas, hacer pruebas con ellas y, en el caso de que se obtuvieran mejoras, incorporarlas al programa.

En resumen, se ha conseguido implementar un reconocedor de habla continua capaz de funcionar correctamente y con una tasa de error aceptable en un entorno controlado, lo cual cumple los objetivos del presente proyecto y puede constituir una base para futuras líneas de desarrollo e investigación.

Bibliografía

[1] X.L. Aubert. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech & Language*, 16(1):89–114, 2002.

[2] J. Benesty, M.M. Sondhi, and Y. Huang. *Springer handbook of speech processing*. Springer Verlag, 2008.

[3] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[4] J. Daciuk, S. Mihov, B.W. Watson, and R.E. Watson. Incremental construction of minimal acyclic finite-state automata. *Computational linguistics*, 26(1):3–16, 2000.

[5] P. Garcia and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.

[6] J.A. Gomez, M.J. Castro, and E. Sanchis. An approach to obtain weighted graphs of words based on phoneme detection. In *9th Conference Speech and Computer*. ISCA, 2004.

[7] J.A. Gómez Adrián, M. Calvo Lance, and E. Sanchis Arnal. Localización de Palabras basada en Grafos de Fonemas. *Procesamiento del Lenguaje Natural*, 44:59–66, 2010.

[8] JE Hopcroft, R. Motwani, and JD Ullman. Introduction to automata theory, languages, and computation, 2001.

[9] F. Jelinek. *Statistical methods for speech recognition*. The MIT Press, 1997.

[10] D. Jurafsky, J.H. Martin, and A. Kehler. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. MIT Press, 2000.

[11] K.F. Lee. *Automatic speech recognition: the development of the SPHINX system*. Kluwer Academic Pub, 1989.

[12] A. Moreno, D. Poch, A. Bonafonte, E. Lleida, J. Llisterri, J.B. Marino, and C. Nadeu. Albayzin speech database: Design of the phonetic corpus. In *Third European Conference on Speech Communication and Technology*. ISCA, 1993.

[13] C. Nadeu Camprubí, J. Llisterri Boix, J.M. Pardo, R. García, A. Rubio San Román, and F. Casacuberta Nolla. Desarrollo de corpus para investigación en tecnologías del habla (Albayzin). *Procesamiento del lenguaje natural*, (12):35, 1992.

[14] D.B. Paul. An efficient A* stack decoder algorithm for continuous speech recognition with a stochastic language model. In *Proceedings of the workshop on Speech and Natural Language*, page 409. Association for Computational Linguistics, 1992.

[15] L. Rabiner and B.H. Juang. Fundamentals of speech recognition. *Englewood Cliffs, NJ*, 1993.

[16] R. Rosenfeld and P. Clarkson. CMU-cambridge statistical language modeling toolkit v2, 1997.

[17] J.M. Sempere and P. García. Inferencia de lenguajes k-explorables en sentido estricto. *Práctica 3 de la asignatura Aprendizaje*, Universidad Politécnica de Valencia, curso 2009-10.

[18] A. Stolcke. SRILM-an extensible language modeling toolkit. In *Seventh International Conference on Spoken Language Processing*, volume 3, pages 901–904. Citeseer, 2002.

[19] I. Torres and A. Varona. k-TSS language models in speech recognition systems. *Computer Speech & Language*, 15(2):127–149, 2001.

[20] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. The HTK book, 2000.