Universidad de La Habana

Facultad de Matemática y Computación

“Detección automática de sonidos de animales empleando Modelos Ocultos de Markov”

Autor: **Osvaldo Alejandro Saez Lombira**

Tutores: **MsC. Alfredo Somoza Moreno**

Dr. Emanuel Mora Macías

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

La Habana, 2015

Agradecimientos

A mis padres, Bárbara y Osvaldo, por todo su apoyo.

A mi novia, Heidy, por su amor sin límites.

A Laura, mi hermana, por su cariño puro e incondicional.

A Mima y Amelia, por apoyarme en los momentos más difíciles.

A Dayán y Rafael por su amistad.

Al profesor Somoza, por enseñarme tanto sobre mí mismo.

Al profesor Emanuel, por siempre estar dispuesto a compartir sus conocimientos.

Osvaldo.

Resumen

En este trabajo se presentan los fundamentos teóricos de los Modelos Ocultos de Markov, así como de los Coeficientes Cepstrales de Mel. Se propone además el uso de Mezclas Gaussianas como la estructura básica en la representación de los estados del modelo y se detallan diversas ideas estocásticas relacionadas con el entrenamiento y reconocimiento de estas. Adicionalmente, estas técnicas son llevadas a la práctica con la implementación de un reconocedor de sonidos de animales al que se le realiza un conjunto de pruebas para probar su eficacia.

Contenido

Abreviaturas

HMM	Modelos Ocultos de Markov
MFCC	Coeficientes Cepstrales de Mel
GMM	Modelo de Mezclas Gaussianas
ASR	Reconocedor Automático de Sonidos
BW	Algoritmo de Baum-Welch
TDF	Transformada Discreta de Fourier
TCD	Transformada Coseno Discreta
FFT	Transformada de Fourier Rápida
ANN	Redes Neuronales Artificiales
KDE	Modelos de Estimación de Densidad del Kernel
SVM	Máquinas de Vectores de Soporte
LPC	Coeficientes de Codificación Predictiva Linear

Capítulo 1: Preliminares

Introducción

El sonido constituye una de nuestras principales formas de comunicación. Usando sus capacidades hemos construido nuestros idiomas y lo hemos empleado por miles de años para analizar el medio que nos rodea. Sin embargo, estas características no son únicas de lo humanos, numerosas especies de animales también lo emplean como medio de comunicación e incluso, muchas de ellas hacen un uso excepcional de él. Por esta razón siempre ha sido el deseo de algunos comprender estas comunicaciones, o al menos identificar a sus protagonistas, por lo que durante mucho tiempo nuestros especialistas han ido descifrando los misterios en ellas, identificando a los animales responsables, y catalogando sus diferentes sonidos acorde a su intención. No obstante, con el incremento en la cantidad de datos a procesar, resulta impracticable el empleo de personas para estos trabajos. Por este motivo los reconocedores automáticos han ido ganando adeptos desde hace ya varias décadas, e incluso desde los años '70s existieron diversos intentos, como se puede constatar en los trabajos de Baker en la Universidad de Carnegie Mellon [1] y Jelinek en IBM [2].

El proceso para realizar la clasificación automática de una muestra sonora pasa por dos fases: entrenamiento y reconocimiento. El primero de estos se caracteriza por la aplicación de un algoritmo que genera, a partir de la señal y de modelo matemático, una instancia específica de este que se "adapta" al sonido en cuestión. Los más utilizados a lo largo de los años han sido: los Modelos Ocultos de Markov (*Hidden Markov Models* o *HMM*) [3], las Redes Neuronales Artificiales (*Artificial Neural Network* o *ANN*) [4], los modelos de Estimación de Densidad del Kernel (*Kernel Density Estimation* o *KDE*) [5], las Máquinas de Vectores de Soporte (*Support Vector Machine* o *SVM*) [5] y las Mezclas Gaussianas (*Gaussian Mixtures Models* o *GMM*) [6]. Todos tienen en común que constituyen un modelo general que se podría especializar si se modificasen los parámetros que lo conforman. Luego, una vez obtenidos los valores que relacionen al modelo con el sonido de interés podemos utilizarlo en la fase de reconocimiento para la identificación de sonidos arbitrarios. En el presente trabajo nos enfocaremos en los Modelos Ocultos de Markov.

Otra cuestión importante a tener en cuenta en el proceso de clasificación es la variabilidad del sonido. Esto se debe a que dos señales, perceptiblemente similares, pueden resultar muy diferentes en su información real. Por esta razón se hace necesaria la inclusión en todos los Sistemas de Reconocimiento Automático (o *ASR* por sus siglas en inglés), de una tecnología de extracción de características como los Coeficientes Cepstrales de las Frecuencias de Mel (*MFCC* por sus siglas en inglés) [7] o los Coeficientes de Codificación Predictiva Linear (*LPC* por sus siglas en inglés). En el presente trabajo se utilizarán los primeros.

Trabajos preliminares

Durante la construcción del ASR para sonidos de animales se han investigado numerosos trabajos que abordan el tema con diversos enfoques. De todos ellos, en mayor o menor medida, se han incorporado ideas. Los siguientes fueron los más representativos.

(Ren, y otros, 2009)

En este trabajo se aplican los HMMs a diferentes especies y tareas bioacústicas usando para cada situación diferentes topologías del modelo. Las labores que se incluyen son: la clasificación del tipo de llamada emitidas por elefantes asiáticos; el reconocimiento de canciones para el Escribano hortelano (Emberiza hortulana), una especie de ave oriunda de Europa; así como una tarea de diferenciación del estrés en las vocalizaciones de las aves de corral para diferentes estímulos. Los resultados ilustran la flexibilidad de los HMMs para varias especies, tipos de vocalizaciones, y tareas de análisis [8].

(Trifa, Kirschel, Taylor, & Vallejo, 2007)

Este artículo explora la habilidad de los HMMS para distinguir entre el canto de cinco especies de aves de la familia Thamnophilidae [9], también conocida como hormigueros, las cuales comparten un mismo territorio en los bosques de México. Se realiza un análisis del impacto en el rendimiento del modelo cuando se utilizan grabaciones de distintas cualidades. Con el empleo de grabaciones limpias el reconocimiento fue casi perfecto, 99.5%. Mientras que para grabaciones con ruido este bajó aproximadamente a un 90%. Además se determinó que el rendimiento también se veía afectado por el tamaño del conjunto de entrenamiento, el método de extracción de características y el número de estados para el modelo.

(Song Scope, 2009)

Este proyecto se enfoca en el producto comercial Song Scope, el cual es un software licenciado que asegura ser capaz de reconocer el canto de un ave a partir de sus vocalizaciones [10]. Esta investigación detalla el desarrollo del software y como logra su identificación usando HMMs y un mecanismo de extracción de características similar a los MFCCs, pero adaptado para ser usado con el canto de aves. Además se utilizan GMMs para los distintos estados del modelo y el clasificador se construye a un nivel silábico[1], en vez del canto íntegro. Acorde a [10] el clasificador fue capaz de identificar el 63% de las vocalizaciones en el conjunto de entrenamiento y el promedio de falsos positivos para los datos de entrenamiento fue de 0.3%. Mientras que para el conjunto de testeo obtuvo un 37% de identificación con un 0.4% de falsos positivos.

(Shaw, 2011)

En este trabajo se realiza una comparación entre varios de los diferentes métodos para el reconocimiento automático de audio, dígase: GMM, HMM, ANN, SVM y una combinación personalizada de ANN/HMM usando como punto de referencia al producto comercial Song Scope. Todo el reconocimiento es realizado en grabaciones completas dado que este método refleja mejor cómo el clasificador será usado en situaciones reales. Los mejores resultados fueron obtenidos por el híbrido ANN/HMM seguido del HMM, ANN y el GMM [4].

¿Qué es el sonido?

El sonido es un fenómeno vibratorio que se transmite en forma de ondas. Para que se genere es necesario que alguna fuente vibre y que exista un medio elástico que pueda transmitirla. Entre los medios de propagación más comunes se encuentran el aire y el agua.

Al sonido se le pueden otorgar ciertas características físicas como la frecuencia, amplitud y tiempo de vibración. En la naturaleza, los sonidos que encontramos se encuentran formados por varias ondas de diferentes frecuencias que nosotros percibimos como una. Estas ondas no tienen todas la misma importancia, por ejemplo, la frecuencia de la onda generada por la fuente vibratoria se le conoce como frecuencia fundamental, mientras aquellas creadas durante el

[1] Esto es para ellos, en el canto del ave, similar a los fonos en el discurso humano.

tránsito de la fundamental a través del medio se les denomina armónicos, y su frecuencia es siempre superior.

Estas características de la onda se traducen en: a mayor (menor) frecuencia fundamental, un sonido más agudo (grave); a mayor (menor) amplitud, más fuerte (débil) y a mayor (menor) tiempo de vibración, más largo (corto).

Sin embargo, el sonido es aún más rico de lo que hemos especificado. La presencia de los armónicos cambia la forma en que lo percibimos y su cantidad e intensidad influyen directamente en lo que denominamos timbre. Característica gracias a la cual podemos identificar a una persona por su voz o a una misma nota, emitida por instrumentos diferentes.

No obstante, todavía no se ha especificado qué es lo que hace a dos sonidos similares sin importar su timbre, frecuencia, intensidad o duración. Después de todo, cuando dos personas diferentes dicen lo mismo, somos capaces de entenderlos por igual. Son nuevamente la frecuencia fundamental y sus armónicos la razón, pues aunque el número e intensidad se mantengan constantes (esto es el timbre), las frecuencias asociadas a cada uno de ellos puede variar, determinando sonidos diferentes.

Coeficientes de Frecuencias Cepstrales de Mel

Los MFCC (por sus siglas en inglés), introducidos por Paul Mermelstein en [11] se han convertido en la actualidad en uno de los métodos más utilizado para la representación del sonido en procesamiento de señales.

La razón de por qué son necesarios estos métodos para representar el sonido, es que los archivos de audio son, en esencia, solo una secuencia numérica que almacena las variaciones de intensidad del sonido a lo largo del tiempo. Esto se debe a que los mecanismos de grabación capturan los cambios en la presión de la onda que transmite el sonido. Por tanto, lo que se obtiene es un archivo que basa su representación en la intensidad y duración y, como sabemos, estas son características muy variables que no permiten identificar de manera precisa su significado[2].

Con el objetivo de solventar esta necesidad los MFCC se inspiran en el sistema auditivo humano e intentan proveer a los modelos de reconocimiento (dígase HMM, en nuestro caso) con una secuencia de valores similar a la que nuestro cerebro recibiría de nuestro oído. Para lograr esto se realizan la siguiente secuencia de pasos:

1. Dividir la señal en fragmentos de corta duración.
2. Calcular la Transformada Discreta de Fourier para cada fragmento.
3. A partir de los valores de la transformada calcular el periodograma que representa la energía para cada valor de frecuencia.
4. Aplicar el banco de filtros de Mel sobre el periodograma y sumar las energías en cada filtro.
5. Tomar el logaritmo de las energías de los filtros.
6. Calcular la Transformada Coseno Discreta de estos valores.
7. Tomar los primeros trece coeficientes de la transformada y abandonar el resto.

Debido a que una señal de audio es muy variable, es prácticamente imposible extraer del sonido completo un grupo de valores estáticos que representen adecuadamente la señal. Sin embargo,

[2] Aunque sí permitirían de manera muy simple reproducir el sonido, ya que solo sería necesario hacer vibrar un nuevo foco emisor (como las bocinas) siguiendo la información numérica almacenada.

a pesar de su variabilidad si tomamos un fragmento lo suficientemente pequeño de ella, podremos asumir que su información es relativamente estática, por lo que se puede extraer información representativa. Por tanto, si dividiésemos el audio original en pequeños fragmentos y para cada uno de ellos obtuviésemos la información que los caracteriza, entonces la secuencia resultante representaría al sonido completo.

En la práctica las duraciones usadas para estos fragmentos oscilan entre 200-400ms, pues para valores más pequeños se contarían con muy pocas muestras para poder extraer información relevante. Además otra práctica muy común es la de superponer estos fragmentos, o sea, el segundo empezaría sobre alguna zona perteneciente al primero y se extendería más allá de esta, alcanzando la misma longitud que el anterior. Esto se hace con el objetivo de lograr cierta continuidad entre los valores obtenidos para cada fragmento. Véase la Fig. 1.

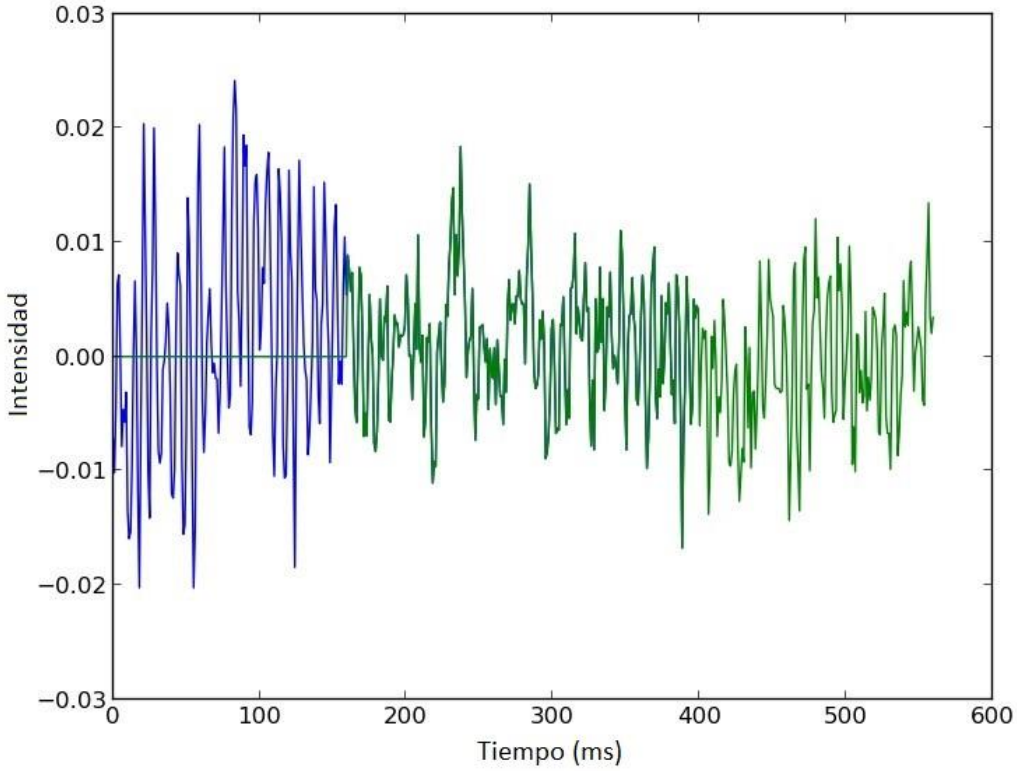

Fig. 1. Dos fragmentos de 400ms superpuestos sobre la señal.

El próximo paso, es probablemente el más importante, ya que describe el proceso mediante el cual se extraen los valores que se consideran significativos del fragmento en cuestión. La justificación formal de por qué se realiza una Transformada de Fourier va más allá del ámbito de este trabajo. No obstante, entiéndase que la Transformada Discreta de Fourier (TDF) es un proceso matemático mediante el cual se extrae de una señal arbitraria las diferentes componentes sinusoidales que la componen junto a sus frecuencias e intensidades [12]. Esto, aplicado sobre una señal sonora extraería las frecuencias e intensidades de los armónicos, que como mencionamos anteriormente son los que conforman el sonido real.

No obstante, la amplitud de las componentes suministrada por la TDF se encuentra en números complejos. Es por eso que en el próximo paso se realiza el cálculo del espectro del periodograma que no es más que la suma de los cuadrados de la parte real e imaginaria pertenecientes a cada

una de las regiones de frecuencia calculadas mediante el FFT[3] para estimar las energías contenidas en cada componente de frecuencia. Sin embargo, estos valores del periodograma todavía contienen información inútil, pues en nuestro sistema auditivo, la cóclea, que es el órgano encargado de identificar las distintas frecuencias presentes en un sonido no es capaz de diferenciar frecuencias muy cercanas entre sí. De hecho para altas frecuencias su habilidad para medir los cambios de frecuencia o su intensidad disminuye, mientras que a frecuencias bajas es lo contrario. Intentando imitar este comportamiento es que surgió la escala de Mel, la cual nos permite agrupar las energías de las frecuencias del periodograma en 26 grupos o regiones donde se combinan las energías de todas las frecuencias que formen parte de dicha región. Esto se logra aplicando el banco de filtros de Mel de la Fig. 3 sobre el espectro del periodograma.

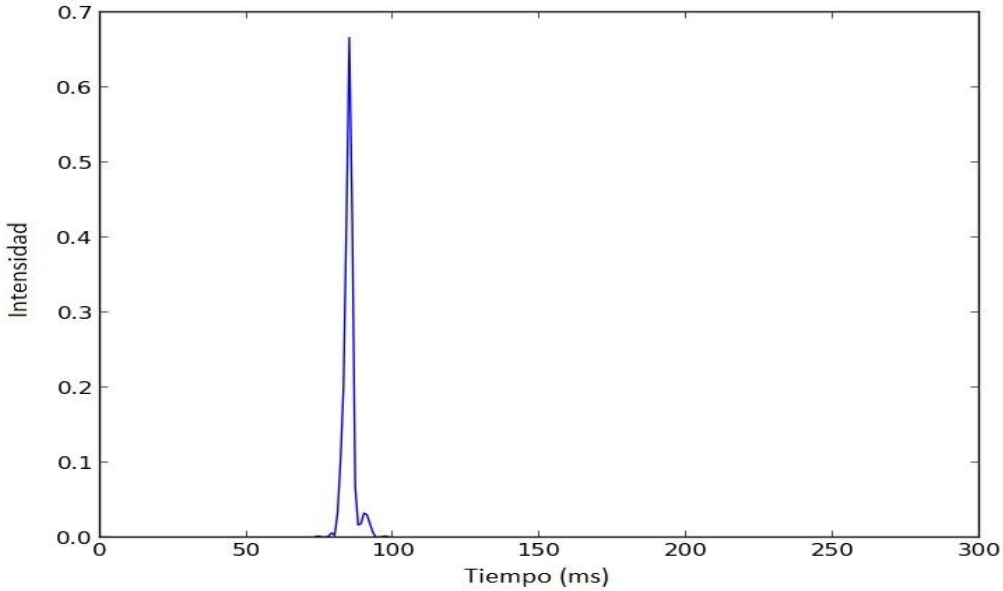

Fig. 2. Un espectro de periodograma donde las frecuencias de mayor energía oscilan de 85-100.

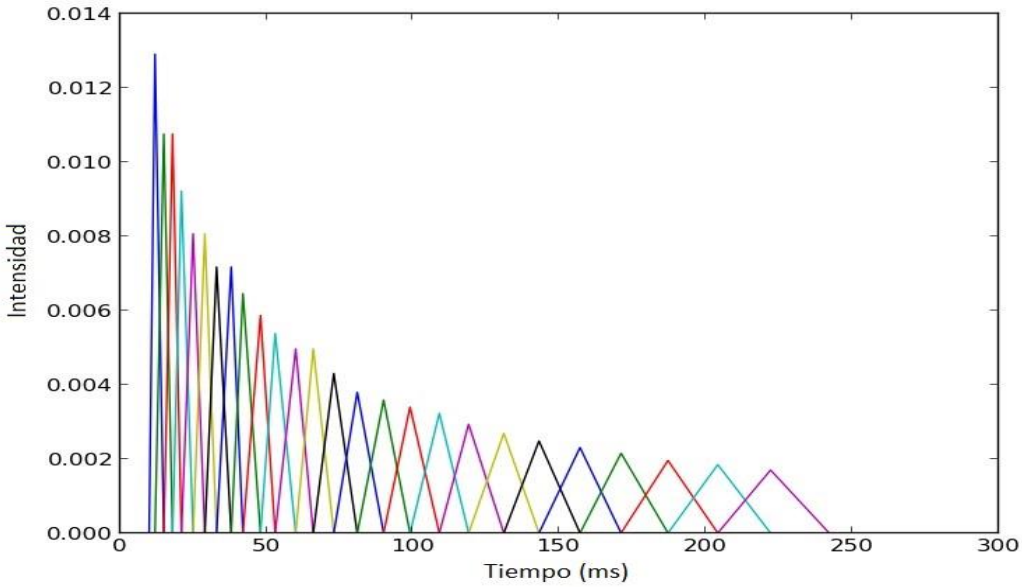

Fig. 3. Filtros de Mel.

[3] Fast Fourier Transform: Algoritmo para calcular la TDF eficientemente.

Al realizar esta operación los 26 valores que se obtienen representan la energía (o intensidad) de las diferentes regiones de frecuencia que se hayan en las componentes del sonido analizado. No obstante, esto aún es insuficiente ya que la energía posee un crecimiento lineal y esto difiere de cómo nuestro oído interpreta las diferentes intensidades de un sonido. De hecho nuestro sistema auditivo expone un crecimiento logarítmico al momento de identificar los cambios de amplitud de los sonidos[4] razón por la cual realizamos en el penúltimo paso la conversión a logaritmos de estos valores.

Por último se le aplican a estos 26 valores logarítmicos la Transformada Coseno Discreta (TCD), la cual realiza en esencia la misma labor que la TFD, pero posee ciertas ventajas que resultan muy útiles para este conjunto de datos. En primer lugar se encuentra el hecho de que los valores obtenidos de los filtros se encuentran muy correlacionados debido al solapamiento de estos y la TCD permite descorrelacionarlos, lo cual nos permitiría utilizar en nuestro HMM matrices de covarianza diagonales. En segundo lugar se encuentra su buena capacidad de compactación lo cual termina ubicando en los primeros coeficientes los valores que representan los cambios más significativos en las energías, mientras que en los últimos 13 mostraría los cambios más veloces. Esto es útil, porque la experiencia ha demostrado que al no incluir los últimos valores se logra un incremento en el rendimiento del ASR [13].

Algo que no incluimos en los pasos, pero que también realizamos, es el proceso de alisamiento de los fragmentos debido a las frecuencias espurias que se generan al aplicar la TDF si los extremos de este son cortados abruptamente [14]. Para evitarlo se multiplica la función que representa los datos del fragmento por la de ventaneo, en nuestro caso la función de Hamming y es este resultado al que se le aplica la TDF. La Fig. 4 muestra, de arriba hacia abajo, la función de Hamming, la señal original del fragmento y la señal alisada.

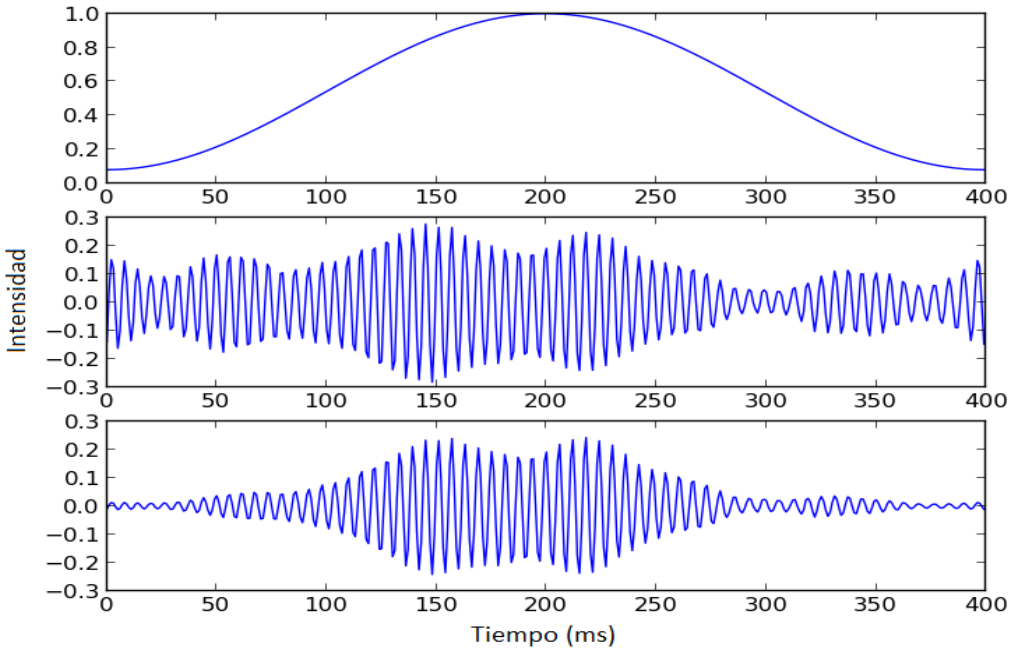

Fig. 4. Ventana de Hamming y su aplicación sobre la señal de audio.

[4] Esta es la razón por la que usamos los decibeles para representar la intensidad de los sonidos, ya que es una medida de crecimiento logarítmico.

Resumen

Hasta aquí se ha planteado la necesidad de utilizar un mecanismo de extracción de características (MFCC) para la correcta interpretación de la señal sonora. Además de que el HMM se ha establecido como el modelo matemático encargado de reconocer la señal auditiva. No obstante, para realizar esta acción los HMM deben, en una primera instancia, entrenarse (Fig. 5) para poder reconocer las características del sonido en cuestión, y después es que podrán ser utilizados para realizar el proceso de reconocimiento (Fig. 6).

Fig. 5. Diagrama del entrenamiento de un Modelo Oculto.

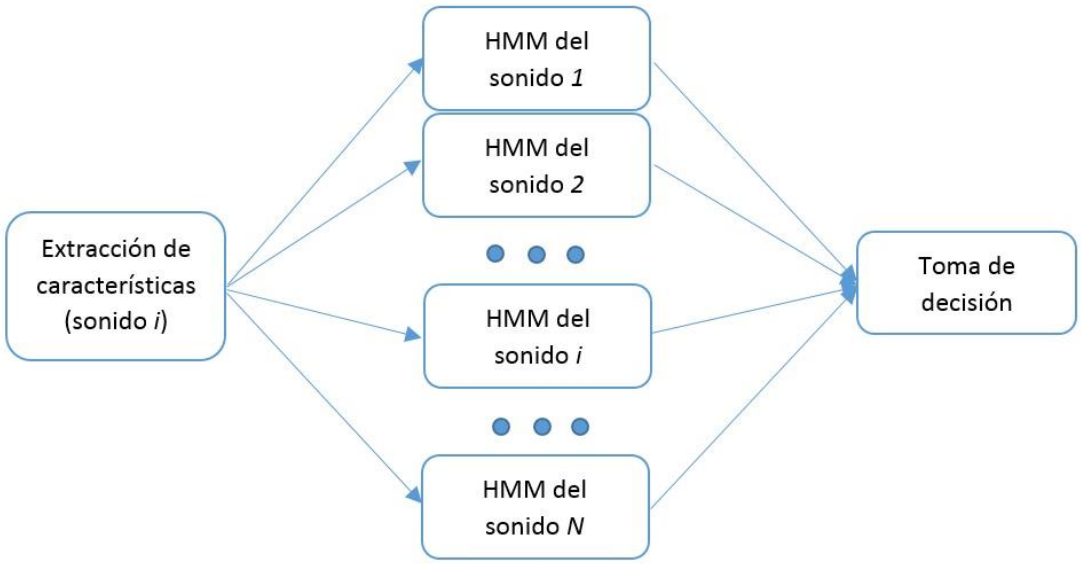

Fig. 6. Diagrama del reconocimiento entre varios Modelos Ocultos.

Veamos con más detalles el fundamento teórico detrás de los Modelos Ocultos de Markov y cómo se pueden utilizar para esta labor.

Capítulo 2: Modelos de Markov

Introducción

La mayor parte de los sistemas que encontramos en la naturaleza se caracterizan por cambiar algunas de sus características cada cierto tiempo. Las razones de por qué realizan esto dependen del sistema y por tanto son muy variadas. No obstante, un hecho común a todos ellos es que dichos cambios pueden ser interpretados como una señal emitida por el proceso a lo largo del tiempo[5], y por tanto una buena manera de estudiar qué es lo que está ocurriendo con el sistema sería estudiar la señal que emite. Si lográsemos entender las señales de estos procesos, entonces podríamos comprenderlos con mayor exactitud permitiéndonos predecir sus acciones, cambiarlas, suprimirlas y simularlas donde también pueda ser útil.

Dichas señales pueden ser discretas o continuas, estacionarias o no estacionarias, puras o corruptas [15]. No obstante, si pudiésemos construir un modelo matemático-computacional adecuado para esta señal, entonces podríamos abstraernos de sus ocurrencias naturales y concentrarnos en dicho modelo para su estudio.

La forma en que suelen ser construidos estos modelos son dos: determinísticos o estocásticos. Los primeros explotan una característica conocida de la señal y utilizan esta información para dar respuestas precisas e invariables dada la entrada. Por otra parte los modelos estocásticos se enfocan en las características estadísticas de la señal y asumen que esta puede ser caracterizada como un proceso estocástico. El Modelo Oculto de Markov (o HMM por sus siglas en inglés) se encuentra dentro de este último grupo.

Los HMM, como su nombre lo indica, se encuentran estrechamente relacionados con la teoría de las cadenas de Markov, la cual debe su nombre al matemático ruso Andrei Markov que las introdujo en 1907. No obstante, no es hasta finales de los 60's y principio de la de los 70's que los trabajos de Leonard E. Baum y sus colegas [3] introdujeron la noción de HMM como una extensión al modelo de las cadenas de Markov.

Independencia condicional

Antes de continuar con los fundamentos de los modelos de Markov es necesario esclarecer el concepto de independencia condicional, pues los resultados que se presentarán en las próximas secciones hacen un uso extensivo de esta.

Intuitivamente, este concepto se refiere a que la probabilidad de ocurrencia de un evento no influye o es influido por el resultado de otro si se sabe que un evento determinado ha ocurrido. Matemáticamente, esto se describe como que dos variables aleatorias X e Y son condicionalmente independientes dado Z, si y solo si

$$P(X, Y|Z) = P(X|Z) * P(Y|Z)$$

Denotaremos esta relación como $X \perp Y|Z$, y para el caso $Z = E$ donde E es el espacio muestral se obtiene una especificación del concepto en la que, para A y B independientes, se cumpliría que $P(A, B) = P(A) * P(B)$.

[5] O cualquier otra variable sobre la que se basen los cambios.

Además de esta definición se puede demostrar que si X e Y son condicionalmente independientes dado Z, entonces las siguientes expresiones siempre se cumplen

$$P(X|Y,Z) = P(X|Z)$$

$$P(Y|X,Z) = P(Y|Z)$$

Probemos este resultado solo para la primera expresión, pues la segunda sería solo invertir el lugar de las variables X e Y.

$$P(X|Y,Z) = \frac{P(X,Y,Z)}{P(Y,Z)} = \frac{P(X,Y|Z) * P(Z)}{P(Y|Z) * P(Z)} = \frac{P(X,Y|Z)}{P(Y|Z)} = \frac{P(X|Z) * P(Y|Z)}{P(Y|Z)} = P(X|Z)$$

El paso de $P(X,Y|Z)$ a $P(X|Z) * P(Y|Z)$ es porque hemos asumido que X e Y son condicionalmente independientes.

Es importante que se tengan en cuentan estos dos resultados. Más adelante se verá que la definición de los HMM se basa en variables aleatorias que poseen relaciones de independencia condicional y muchas veces explotaremos este hecho usando los resultados expuestos aquí.

Procesos estocásticos

Antes se ha mencionado el concepto de los procesos estocásticos (o aleatorios), pero no se especificó en qué consiste. En la teoría de la probabilidad el concepto de variable aleatoria corresponde al de una función con dominio en el conjunto de los posibles resultados de un problema aleatorio e imagen en los números reales, intuitivamente, estos números se asignan a los posibles resultados de manera que denoten la ocurrencia de un determinado evento. Relacionado con toda variable aleatoria se encuentra el concepto de distribución de probabilidad: otra función que toma como dominio los valores de la variable, mientras su imagen es igual a la probabilidad (o densidad para variables aleatorias continuas) de la ocurrencia de esta.

Dicho esto un proceso aleatorio no es más que un conjunto ordenado de variables aleatorias. O sea que estas evolucionan en función de otra variable, por lo general el tiempo. Esto es

$$Conjunto\ de\ X_t\ donde\ t \in T\ con\ T \subseteq \mathbb{R}$$

En estos procesos la distribución de probabilidad puede ser la misma para cada una de las variables aleatorias o no; así como la dependencia entre estas variables, o sea cómo el valor de una afecta a otras.

Si se asumen a partir de ahora procesos aleatorios discretos (o sea $T = \mathbb{N} \subseteq \mathbb{R}$), entonces un proceso típico es el independiente e idénticamente distribuido (i.i.d) en el cual todas las variables aleatorias poseen la misma distribución de probabilidad y son independientes entre sí. Por tanto la probabilidad de la ocurrencia de una secuencia de variables aleatorias: $\{X_1, X_2, \dots, X_n\}$ es

$$P(X_1, X_2, \dots, X_n) = P(X_1)P(X_2) \dots P(X_n)$$

Este proceso es realmente conveniente debido a su simplicidad. No obstante, muchos sistemas reales no se comportan de esta manera, en la mayoría existe algún tipo de dependencia en el pasado, rompiendo la asunción de independencia, la cual resultaría inexacta y contraproducente.

Debido a esto se han investigado un grupo de procesos estocásticos, entre ellos el llamado proceso de Markov de orden k donde la próxima variable aleatoria depende solo de las k anteriores. Esto significa que

$$P(X_n|X_{n-1}, X_{n-1}, \dots, X_1) = P(X_n|X_{n-1}, X_{n-2}, \dots, X_{n-k})$$

A este resultado se le conoce como propiedad de Markov, y lo que significa es que $X_{n-k-1}, X_{n-k-2}, \dots, X_1$ son condicionalmente independientes de X_n dado $X_{n-1}, X_{n-2}, \dots, X_{n-k}$. Por tanto este proceso resuelve la limitación del i.i.d y modela dependencia en el pasado. Luego

$$\begin{aligned} P(X_1, X_2, \dots, X_n) &= P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1}, \dots, X_1) \\ &= P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1}|X_{n-2}, \dots, X_1)P(X_{n-2}, \dots, X_1) \end{aligned}$$

Si realizamos este proceso de sustituir la probabilidad conjunta usando su despeje de la definición de probabilidad condicional $n-2$ veces más, entonces obtenemos que

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1}|X_{n-2}, \dots, X_1) \dots P(X_2|X_1)P(X_1)$$

De donde, si asumimos la propiedad Markov con orden k para este proceso de Markov

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_{n-k})P(X_{n-1}|X_{n-2}, \dots, X_{n-1-k}) \dots P(X_2|X_1)P(X_1)$$

Este resultado para la probabilidad conjunta de un proceso de Markov le permite modelar eficientemente la dependencia en el pasado.

Cadenas de Markov

El término cadena de Markov se usa indistintamente dependiendo de la fuente. No obstante, su uso más común, y el que se le dará aquí, es aquel donde denota un proceso de Markov discreto como el que se trata al final de la sección anterior, en el que además las variables aleatorias toman valores en un conjunto discreto y finito.

Una interpretación útil de las cadenas de Markov es aquella donde se consideran a los posibles valores que pueden tomar las variables aleatorias que conforman el proceso como posibles estados en los que se encontraría el sistema. Declaremos dicho conjunto como $S = \{S_1, S_2, \dots, S_N\}$, siendo N el número de estados.

Como se ha dicho anteriormente todo proceso de Markov, y por tanto, toda cadena de Markov debe cumplir que las variables aleatorias que la componen cumplan con la propiedad de Markov para algún orden k.

Obsérvese que la elección del valor k puede resultar complicada ya que al incrementar su valor, con el objetivo de asegurar que el modelo represente correctamente las dependencias que existen con el pasado, se vuelve más complicado obtener el valor de cualquier probabilidad conjunta. Por ejemplo para un momento n del proceso tenemos

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_1)P(X_{n-1}|X_{n-2}, \dots, X_1) \dots P(X_2|X_1)P(X_1)$$

Aplicando la propiedad de Markov con orden $k = 4$ se obtiene

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1}, \dots, X_{n-4})P(X_{n-1}|X_{n-2}, \dots, X_{n-5}) \dots P(X_2|X_1)P(X_1)$$

Sin embargo para $k = 1$

$$P(X_1, X_2, \dots, X_n) = P(X_n|X_{n-1})P(X_{n-1}|X_{n-2}) \dots P(X_2|X_1)P(X_1)$$

Claramente a menor orden más simple se vuelve este cálculo, sin embargo a menor orden se pierde mayor exactitud en el modelado de procesos dependientes del pasado distante. Al parecer nos encontramos en un impasse donde debemos elegir entre exactitud o facilidad de uso: esto no es del todo cierto.

Tomando una cadena de Markov $X_1, \dots, X_n$ con orden $k > 1$, entonces $P(X_n|X_{n-1}, \dots, X_{n-k})$ denota las dependencias de la variable X_n en el pasado. Si declaramos las variables aleatorias X'_{n-i} donde $i \geq 0$ tal que $X'_{n-i} = X_{n-i} \cap \dots \cap X_{n-i-k}$ (nótese que $X'_n = X_n \cap \dots \cap X_{n-k}$ y que $X'_{n-1} = X_{n-1} \cap \dots \cap X_{n-1-k}$) entonces

$$P(X_n|X_{n-1}, \dots, X_{n-k}) = P(X_n|X_{n-1}, \dots, X_{n-k}, X_{n-1-k})$$

Obsérvese que podemos incluir X_{n-1-k}, porque es condicionalmente independiente de X_n dado $X_{n-1}, \dots, X_{n-k}$ por la propiedad de Markov. A continuación denotamos a $X_{n-1}, \dots, X_{n-k}$ como A y probemos que $P(X_n|A, X_{n-1-k}) = P(X_n, A|A, X_{n-1-k})$.

$$P(X_n, A|A, X_{n-1-k}) = \frac{P(X_n, A, A, X_{n-1-k})}{P(A, X_{n-1-k})} = \frac{P(X_n, A, X_{n-1-k})}{P(A, X_{n-1-k})} = P(X_n|A, X_{n-1-k})$$

Luego $P(X_n|A, X_{n-1-k}) = P(X_n, A|A, X_{n-1-k})$ es lo mismo que

$$P(X_n|X_{n-1}, \dots, X_{n-k}, X_{n-1-k}) = P(X_n, X_{n-1}, \dots, X_{n-k}|X_{n-1}, \dots, X_{n-k}, X_{n-1-k})$$
$$= P(X'_n|X'_{n-1})$$

De donde $P(X_n|X_{n-1}, \dots, X_{n-k}) = P(X'_n|X'_{n-1})$.

Este resultado demuestra que para toda cadena de Markov de orden k existe otra equivalente de orden 1. Luego, no es necesario elegir el orden ya que se puede usar $k = 1$, siempre y cuando se incluyan suficientes estados sobre los cuales la variable pueda tomar valores ya que si la cadena X_n (de orden $k > 1$) posee N estados entonces se cumple que el número necesario de valores para X'_n (la cadena de orden 1 equivalente) es N^k.

Considerando la probabilidad conjunta de una cadena de Markov de orden $k = 1$ se tiene que

$$P(X_1 = q_1, \dots, X_n = q_n) = P(X_n = q_n|X_{n-1} = q_{n-1}) \dots P(X_2 = q_2|X_1 = q_1)P(X_1 = q_1)$$

Donde q_i, $1 \leq i \leq n$ es el estado que ocurrió en el momento i. Los términos $P(X_n = q_j|X_{n-1} = q_i) = a_{ij}$, donde $1 \leq i, j \leq N$, se refieren a la probabilidad de transición del estado i al j en el momento $n - 1$ y el término $P(X_1 = q_1) = \pi_i$, $1 \leq i \leq N$ denota la probabilidad de que el proceso empiece por el estado q_1 (obsérvese que q_1 es cualquier elemento de S).

Si en el caso de la probabilidad de transición a_{ij} para todo par i, j sucede que su valor no cambia en relación al tiempo esto es

$$P(X_{n'} = q_j|X_{n'-1} = q_i) = P(X_{n''} = q_j|X_{n''-1} = q_i)$$

Con $n' \neq n''$, entonces a la cadena de Markov se le denomina homogénea. De ser así, se puede formar una matriz A_{NxN}, donde sus entradas (i, j) son iguales a a_{ij}. A esta se le denomina matriz de transiciones. Un par de propiedades importante que cumple esta matriz de transición, o más específicamente, las probabilidades de transición es que

$$\sum_{j=1}^{N} a_{ij} = 1\,, 1 \leq i \leq N$$

$$a_{ij} \geq 0, 1 \leq i, j \leq N$$

Así como también para el caso de $\pi_i, 1 \leq i \leq N$

$$\sum_{i=1}^{N} \pi_i = 1$$

$$\pi_i \geq 0, 1 \leq i \leq N$$

Ambos pares de restricciones responden a la necesidad de que se cumplan las restricciones de la teoría de la probabilidad para los valores del modelo.

Para ver un ejemplo práctico de todo lo que se ha discutido hasta ahora consideremos el problema de predecir el tiempo. Obsérvese que este problema es muy adecuado para ser modelado mediante una cadena de Markov ya que el estado del tiempo para un día determinado no es un fenómeno determinista, sino aleatorio. Por lo que para modelar una secuencia de días y sus diferentes estados del tiempo se necesitará una secuencia de variables aleatorias donde cada una corresponda a la probabilidad de ocurrencia de cualquier estado del tiempo ese día específico. Además, el tiempo es un fenómeno que claramente muestra relaciones con el pasado; ya que, por ejemplo, estadísticamente podemos constatar que la probabilidad de dos días continuos de sol es más alta que la de un día de sol seguido por otro de lluvia. Con esto se cumplen todas las condiciones necesarias para aplicar satisfactoriamente las cadenas de Markov. Veamos el modelo.

$$S = \{S_1 = lluvioso,\ S_2 = nublado,\ S_3 = soleado\}$$

Donde S denota los diferentes estados del tiempo y además sean, arbitrariamente[6], las probabilidades de transición de uno de estos estados al próximo

$$A = \begin{pmatrix} 0.4 & 0.3 & 0.3 \\ 0.2 & 0.6 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix}$$

Por último, los valores del vector de distribución inicial son $\pi = (\pi_1, \pi_2, \pi_3)$ que denotan las probabilidades de empezar por alguno de los estados posibles. En este problema, como no se posee ninguna información concerniente al momento de inicio del análisis, asúmase que las probabilidades de inicio son $\pi = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

Con esta información, y asumiendo que el modelo representa correctamente el problema a tratar, entonces se puede utilizar esta cadena de Markov para generar una simulación del proceso natural subyacente y de ahí inferir conclusiones acerca de él. Por ejemplo, si se intentase responder a la pregunta de cuál es la probabilidad de que suceda una secuencia de 8 días donde el estado del tiempo se comporte de la siguiente manera: $S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3$, se necesitaría obtener la probabilidad con la que el modelo (formado por los datos de A y π anteriores) genera dicha secuencia y, dado que se asumió que este era una buena

[6] Más adelante veremos cómo hacer esto de manera que refleje la realidad observada.

representación del tiempo, entonces podremos admitir su resultado como repuesta a la pregunta.

$$P(O|Modelo) = P(S_3, S_3, S_3, S_1, S_1, S_3, S_2, S_3|Modelo)$$

$$= P(S_3|S_3) * P(S_3|S_3) * P(S_3|S_1) * P(S_1|S_1) * P(S_1|S_3) * P(S_3|S_2) * P(S_2|S_3) * P(S_3)$$

$$= \pi_3 * a_{33} * a_{33} * a_{31} * a_{11} * a_{13} * a_{32} * a_{23}$$

$$= \frac{1}{3} * (0.8) * (0.8) * (0.1) * (0.4) * (0.3) * (0.1) * (0.2)$$

$$= 5{,}12 * 10^{-5}$$

Extensión a los HMM

Modelos como el de las cadenas de Markov, aunque comunes, presentan una importante deficiencia: el número finito de estados que puede tomar el sistema genera inflexibilidad. De hecho, el problema anterior es una evidencia clara de esto ya que el estado del tiempo es mucho más rico de como ha sido declarado. Los días no son nunca completamente soleados, lluviosos o nublados. Si somos rigurosos, un día no luce como los estados que se han establecido, en su lugar lo que se observa es un grupo de características que pueden ser distinguidas como un estado específico, es por eso que aunque dos días sean catalogados de manera similar sus características son diferentes. O sea, en cada momento existe un estado que se alza como favorito, sin embargo, esta "opinión" no es absoluta, ya que no es a él al que observamos, sino a las características del día que lo sugieren, y por tanto una "opinión" diferente puede catalogar ese mismo momento con un estado completamente distinto.

Dicho esto, el problema de la sección anterior cambiaría. Ahora en lugar de preguntar directamente por la secuencia de estados, se utilizarían las características del día en que estamos interesados las cuales son más exactas e imparciales. Para esto definimos estas características como vectores de $\mathbb{R}^3$ donde cada componente corresponde, en este orden, con la humedad, temperatura y vientos promedios de día. Por tanto, la secuencia de la sección anterior podría lucir de la siguiente manera:

$$O = \begin{Bmatrix} (20, 33, 15), (22, 34, 16), (26, 33, 17), (75, 28, 22), \\ (77, 27, 25), (40, 31, 16), (55, 30, 19), (48, 32, 17) \end{Bmatrix}$$

La respuesta, no obstante, a $P(O|Modelo)$ presenta una mayor complejidad que la de la sección anterior. Es por esta razón que se retomará más adelante.

En definitiva, en estos tipos de procesos donde la existencia de estados es obvia, pero no visible, ya que todo lo que se obtiene del sistema son observaciones distintas que solo sugieren la presencia de estos, es cuando el uso de los HMM es prácticamente indispensable. La Fig. 7 propone una interpretación muy intuitiva de los HMM donde los estados corresponden a urnas y las observaciones a bolas de colores que se encuentran dentro de estas, por lo que la probabilidad de que dentro de una urna se elija una bola específica corresponde a la probabilidad de que tal estado emita esa observación. Esta interpretación posee la ventaja de que puede degenerarse (de manera muy simple) en una cadena de Markov ya que si cada urna contiene solo una bola esta se convierte en la única elección, y por tanto, es igual al caso donde el estado es directamente observable.

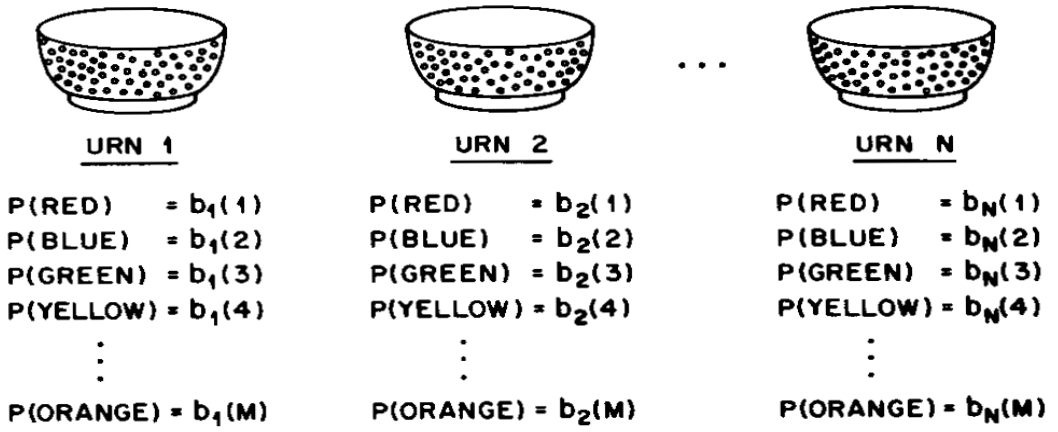

Fig. 7. Un modelo de bolas y urnas con N estados que muestra el procedimiento general utilizado en los HMM's para el caso discreto. Ilustración tomada de [15].

Este sistema, aunque simple, puede modelar correctamente la situación que se nos había presentado con el problema del estado del tiempo. Ya que las urnas representarían los posibles estados del día, mientras que las bolas corresponderían a las observaciones posibles para ese estado del tiempo (las cuales serían los vectores en $\mathbb{R}^3$ antes mencionados). Esta interpretación saca a relucir las partes componentes de un HMM:

- N, el número de estados del modelo. Los cuales cumplen la misma función que se ha descrito en la sección anterior, pero ahora se encuentran ocultos. Los estados se denotan como $S = \{S_1, S_2, \dots, S_N\}$, mientras que al estado en el momento t del proceso se le denota por q_t.
- A, la matriz de probabilidad de transición entre estados que ya también ha sido descrita en la sección anterior.
- π, el vector de probabilidad inicial para todos los estados.
- M, el número de observaciones discretas y distintas por cada estado. Este es un elemento que no siempre está presente, ya que como se verá más adelante, muchas veces se posee un infinito número de observaciones por estado (como en el ejemplo del estado del tiempo). Denotamos el conjunto de observaciones como V.
- $B = b_j(O)$, función de probabilidad que denota la posibilidad de ocurrencia de cualquier observación en el estado $j, 1 \leq j \leq N$. Por citar un ejemplo, sea $v_t \in V$ la observación que ocurren en el momento t, entonces se cumple que $b_j(v_t) = P(v_t | q_t = S_j)$.

De forma similar a como se hizo con las cadenas de Markov, si se establecen valores para todas estas incógnitas (N, A, π, M, B) y se desea generar una secuencia de observaciones[7] $\{O_1, \dots O_T\}$ entonces se sigue la siguiente secuencia de pasos (de la cual más tarde veremos su relevancia matemática):

1. Escoger q_1 de acuerdo al vector de probabilidad inicial.
2. Establecer $t = 1$, donde t es el tiempo.
3. Elegir $O_t \in V$ de acuerdo a $b_{q_t}(O_t)$.

[7] Nótese la diferencia con las cadenas de Markov que generan secuencias de estados.

4. Transitar a un estado q_{t+1} de acuerdo a la probabilidad de transición desde el estado q_t.
5. Establecer $t = t + 1$ y retornar al paso 3 si $t < T$. Si no salir.

Observe el diagrama de la Fig. 8 para tener una idea de lo que significa la definición anterior.

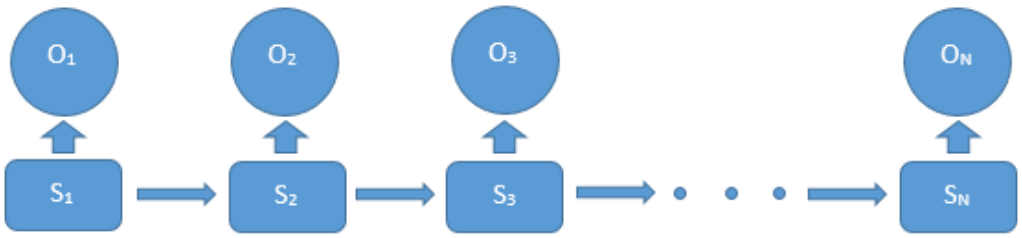

Fig. 8. Diagrama que ilustra el funcionamiento de un HMM.

Matemáticamente, un HMM se describe como dos procesos estocásticos discretos: el de las observaciones $O_1, O_2, \dots, O_T$, cuyas variables aleatorias toman valores en $\mathbb{R}^n$, y el de los estados $X_1, X_2, \dots, X_T$ que toman valores en un conjunto finito y discreto. Cuyas variables aleatorias mantienen las siguientes restricciones:

(1) $$\{X_{t:T}, O_{t:T}\} \perp \{X_{1:t-2}, O_{1:t-1}\} | X_{t-1}$$

(2) $$O_t \perp \{O_{\bar{t}}, X_{\bar{t}}\} | X_t$$

Donde $\perp$ representa independencia condicional y $\bar{t}$ es el complemento de t, con $1 \leq t \leq T$. Ambas definiciones son equivalentes, no obstante, nos apoyaremos en la naturaleza más formal de esta última para nuestras demostraciones.

Antes, en el problema del tiempo, se ha dejado en pausa el cálculo de $P(O|Modelo)$ donde O constituía una secuencia de observaciones de varios días y $Modelo$ hace referencia al uso, en dicho cálculo, de los parámetros que se han establecido para los componentes del HMM. Para lograr este objetivo consideremos

$$P(X_{1:T}, O_{1:T}) = P(X_T, \dots, X_1, O_T, \dots, O_1)$$
$$= P(O_T|X_T, \dots, X_1, O_{T-1}, \dots, O_1) * P(X_T|X_{T-1}, \dots, X_1, O_{T-1}, \dots, O_1)$$
$$* P(X_{T-1}, \dots, X_1, O_{T-1}, \dots, O_1)$$

Aplicando la independencia condicional definida en (2) sobre el primer factor y la de (1) en el segundo se obtiene

$$= P(O_T|X_T) * P(X_T|X_{T-1}) * P(X_{T-1}, \dots, X_1, O_{T-1}, \dots, O_1)$$

$$\dots$$

(3) $$= P(X_1) * P(O_1|X_1) * \prod_{i=2}^{T} P(X_i|X_{i-1}) * P(O_i|X_i)$$

Siguiendo con nuestro objetivo de calcular $P(O|Modelo)$, observemos que $P(X_{1:T}, O_{1:T})$ no es más que la probabilidad de obtener una secuencia de observaciones específica, pero limitada a una secuencia de estados específica que la genere. Luego, si buscamos la probabilidad de esta secuencia de observaciones sin importar la secuencia de estados entonces necesitaremos marginalizar esta probabilidad. Por tanto

$$P(O|Modelo) = \sum_{x_{i:T}} P(x_{i:T}, O_{1:T})$$

Analizando los factores de (3), se puede observar que $P(O_i|X_i) = b_{q_i}(O_i)$, $P(X_i) = \pi_i$ y $P(X_i|X_{i-1}) = a_{q_{i-1}q_i}$. Luego, si se tuvieran todos estos factores se podría calcular esta probabilidad.

Problemas fundamentales de los HMM

Vista la definición de los HMM, existen tres problemas básicos que deben ser resueltos para permitir el uso de estos en aplicaciones prácticas.

1. **Problema 1:** Dada la secuencia de observaciones $O = O_1, O_2, \dots, O_T$, producto de una ocurrencia de algún proceso estocástico real o simulado y $\lambda = (A, B, \pi)$[8] es de interés calcular eficientemente $P(O|\lambda)$[9]. O sea la probabilidad de que el modelo λ haya emitido a O.
2. **Problema 2:** Dada la secuencia de observaciones $O = O_1, O_2, \dots, O_T$ y el modelo λ cómo encontramos la secuencia de estados $Q = q_1, q_2, \dots, q_T$ que mejor explica la secuencia de observaciones O para el modelo.
3. **Problema 3:** Dada la secuencia de observaciones $O = O_1, O_2, \dots, O_T$ cómo calculamos los parámetros para $\lambda = (A, B, \pi)$ tal que se maximice $P(O|\lambda)$.

El **problema 1** se le conoce como el problema de evaluación donde dado un modelo y una secuencia de observaciones se intenta calcular cuál es la probabilidad de que dicho modelo haya emitido esa secuencia. Además puede ser interpretado como un problema de puntaje donde se posee una secuencia de observaciones de origen desconocido y se tiene un determinado número de modelos, por lo que si se resuelve este problema para cada uno de ellos se puede utilizarse esta información para determinar cual explica mejor la secuencia y asignarle un significado a esta acorde al modelo.

Por ejemplo, imagínese que se tiene una secuencia de observaciones O que representa el sonido de una letra del abecedario emitida por un individuo particular. Además, existen un conjunto de modelos construidos, dígase $\lambda_A, \lambda_B, \dots, \lambda_Z$ listos para reconocer cada uno de ellos una letra específica. Luego, tomando $argmax_{\alpha \in \{A,B,\dots,Z\}}(P(O|\lambda_\alpha))$ podemos responder a la pregunta de qué letra ha dicho el individuo.

El **problema 2** es aquel donde intentamos "descubrir" la parte oculta del HMM. Razón por la cual no se puede establecer una respuesta correcta para este problema ya que es imposible determinar, en primera instancia, si los estados que hemos elegido coinciden con los del proceso real o si las transiciones y emisiones son fidedignas a este. Todo esto, sumado al hecho de que las observaciones pueden ser emitidas por cualquier estado (aunque con distintas probabilidades) provoca que la respuesta a este problema dependa del criterio que se utilice para decidir qué estado asignar en cada momento a la secuencia de observaciones.

El **problema 3** es probablemente el más importante de los tres. Es aquel que permite obtener los parámetros del modelo que mejor resuelven el problema 1 para una secuencia de observaciones determinada. O sea, sin una respuesta para él no se contaría con modelos adecuados para explicar ningún proceso. La secuencia de observaciones para obtener al modelo se le conoce como secuencia de entrenamiento, ya que es utilizada para "entrenar" al modelo.

Veamos ahora soluciones matemáticas formales a cada uno de estos problemas.

---

[8] Así denominaremos a (N, A, π, M, B).

[9] Antes lo hemos visto como $P(O|Modelo)$.

Solución al problema 1

Se desea calcular la probabilidad de que la secuencia de observaciones $O = O_1, O_2, \dots, O_T$ ocurra a partir del modelo λ, esto es $P(O|\lambda)$. La manera más simple de lograrlo es enumerando todas las posibles secuencias de estados de longitud T y calcular, en cada una de ellas, la probabilidad de que generen a O para luego unificar todas estas probabilidades. Dicho esto sea $Q = q_1, q_2, \dots, q_T$ una secuencia de estados específica, entonces

$$P(O|\lambda) = \sum_{Q \in K} P(O, Q|\lambda) \quad (4)$$

Donde K es el conjunto de todas las posibles secuencias de estados de longitud T. A su vez

$$P(O, Q|\lambda) = \frac{P(O, Q, \lambda)}{P(\lambda)}$$

$$= \frac{P(O|Q, \lambda) * P(Q, \lambda)}{P(\lambda)}$$

$$= \frac{P(O|Q, \lambda) * P(Q|\lambda) * P(\lambda)}{P(\lambda)}$$

$$= P(O|Q, \lambda) * P(Q|\lambda) \quad (5)$$

En definitiva el valor que se busca depende de la probabilidad de que la secuencia analizada O sea generada por una secuencia dada de estados específica, así como de la probabilidad de que tal secuencia de estados ocurra. Luego

$$P(O|Q, \lambda) = P(O_1, O_2, \dots, O_T|q_1, q_2, \dots, q_T, \lambda)$$

$$= \frac{P(O_1, O_2, \dots, O_T, q_1, q_2, \dots, q_T, \lambda)}{P(q_1, q_2, \dots, q_T, \lambda)}$$

$$= \frac{P(O_1|q_1, \dots, q_T, O_2, \dots, O_T, \lambda) * P(q_1, \dots, q_T, O_2, \dots, O_T, \lambda)}{P(q_1, q_2, \dots, q_T, \lambda)}$$

$$= \frac{P(O_1|q_1, \dots, q_T, O_2, \dots, O_T, \lambda) * P(O_2|q_1, \dots, q_T, O_3, \dots, O_T, \lambda) * P(q_1, \dots, q_T, O_3, \dots, O_T, \lambda)}{P(q_1, q_2, \dots, q_T, \lambda)}$$

…

$$= \frac{P(O_1|q_1, \dots, q_T, O_2, \dots, O_T, \lambda) * P(O_2|q_1, \dots, q_T, O_3, \dots, O_T, \lambda) * \dots * P(O_T|q_1, \dots, q_T, \lambda) * P(q_1, \dots, q_T, \lambda)}{P(q_1, q_2, \dots, q_T, \lambda)}$$

$$= P(O_1|q_1, \dots, q_T, O_2, \dots, O_T, \lambda) * P(O_2|q_1, \dots, q_T, O_3, \dots, O_T, \lambda) * \dots * P(O_T|q_1, \dots, q_T, \lambda)$$

Aplicando la propiedad (2) de la definición de los HMM obtenemos

$$P(O|Q, \lambda) = P(O_1|q_1) * P(O_2|q_2) * \dots * P(O_T|q_T) = b_{q_1}(O_1) * b_{q_2}(O_2) * \dots * b_{q_T}(O_T)$$

Lo cual es calculable a partir de los componentes de una HMM. No obstante, volviendo a (5) todavía queda por analizar cómo obtener $P(Q|\lambda)$.

$$P(Q|\lambda) = P(q_1, q_2, \dots, q_T|\lambda)$$

$$= \frac{P(q_1, q_2, \dots, q_T, \lambda)}{P(\lambda)}$$

$$= \frac{P(q_T|q_{T-1}, q_{T-2} \dots, q_1, \lambda) * P(q_{T-1}, \dots, q_1, \lambda)}{P(\lambda)}$$

$$= \frac{P(q_T|q_{T-1},\dots,q_1,\lambda) * P(q_{T-1}|q_{T-2},\dots,q_1,\lambda) * P(q_{T-2},\dots,q_1,\lambda)}{P(\lambda)}$$

$$= \frac{P(q_T|q_{T-1},\dots,q_1,\lambda) * P(q_{T-1}|q_{T-2},\dots,q_1,\lambda) * \dots * P(q_1|\lambda) * P(\lambda)}{P(\lambda)}$$

$$= P(q_T|q_{T-1},\dots,q_1,\lambda) * P(q_{T-1}|q_{T-2},\dots,q_1,\lambda) * \dots * P(q_1|\lambda)$$

Aplicando la propiedad (1) de la definición de los HMM se obtiene

$$P(Q|\lambda) = P(q_1|\lambda) * P(q_2|q_1) * \dots * P(q_{T-1}|q_{T-2}) * P(q_T|q_{T-1})$$

$$P(Q|\lambda) = \pi_{q_1} * a_{q_1 q_2} * \dots * a_{q_{T-2} q_{T-1}} * a_{q_{T-1} q_T}$$

Resultando, al igual que la probabilidad anterior, en un producto de valores conocidos para el modelo. Luego debido a (5)

$$P(O,Q|\lambda) = \pi_{q_1} * P(O_1|q_1) * a_{q_1 q_2} * P(O_2|q_2) * \dots * a_{q_{T-1} q_T} P(O_T|q_T)$$

Por lo que volviendo a (4)

$$P(O|\lambda) = \sum_{\{q_1,\dots,q_T\} \in K} \pi_{q_1} * P(O_1|q_1) * a_{q_1 q_2} * P(O_2|q_2) * \dots * a_{q_{T-1} q_T} P(O_T|q_T)$$

Interpretando la expresión anterior nos muestra que por cada posible secuencia primeramente se calcula la probabilidad de que el primer estado de esta sea el estado inicial del proceso y de que la primera observación suceda en ese estado. Luego se incrementa el tiempo y se realiza una transición del estado anterior al presente para luego obtener la probabilidad de que la observación ocurra en ese estado. Este proceso se repite hasta llegar al final de la cadena y luego se realiza todo de nuevo para la próxima secuencia de estados.

Es obvio que resolver el problema 1 de esta manera conllevaría un costo computacional enorme. Pues el número posible de secuencias de estados de longitud T es N^T, donde N es el número de estados; y por cada una de ellas se realizan dos productos de longitud T (estos son $P(O|Q,\lambda)$ y $P(Q|\lambda)$). Entonces, el costo total de ejecución es de alrededor $2TN^T$, lo cual es impráctico. Afortunadamente, existe un procedimiento mucho más eficiente.

El **Forward-Backward Algorithm** [16] [17] no es exactamente un único algoritmo ya que consiste de dos sub-procedimientos: uno para calcular las variables *forward* y otro para calcular las *backward*, siendo utilizadas ambas en multitud de resultados relacionados con los HMM. No obstante, es la variable *forward* la que presenta una gran utilidad para el problema en cuestión. Para entenderla defínase el siguiente valor probabilístico

$$\alpha_t(i) = P(O_1,\dots,O_t,q_t = S_i|\lambda)$$

El cual se definirá como la variable *forward* para el estado i en el momento t. El valor de esta probabilidad denota la posibilidad de que el modelo λ haya generado las primeras t observaciones, sin importar los estados usados, excepto por el del momento t que debe ser S_i.

Con esta interpretación en mente considérese la variable *forward* $\alpha_T(i)$, la cual es la probabilidad de que el modelo genere toda la secuencia de observaciones (recuérdese que esta tiene longitud T) y de que el último estado sea i, lo cual sería $P(O,q_T = S_i|\lambda)$. Luego, como la secuencia de observaciones debe culminar en alguno de los N estados, se tendría que

$$(6) \qquad P(O|\lambda) = \sum_{i=1}^{N} P(O,q_T = S_i|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

Por lo que las variables $\alpha_T(i)$ para todo $1 \leq \text{i} \leq \text{N}$ también pueden resolver el problema 1. Sin embargo, a diferencia de la solución anterior, esto se puede lograr eficientemente pues las variables *forward* son recursivas y, como se verá a continuación esto nos permitirá reducir el volumen de cálculos usando programación dinámica.

Primeramente, obsérvese cómo se calcula $\alpha_1(i)$, la cual es la probabilidad de que el estado en el momento inicial sea i, así como que en él se emita la observación O_1. Por lo que

$$\alpha_1(i) = P(O_1, q_1 = S_i|\lambda)$$

$$= \frac{P(O_1, q_1 = S_i, \lambda)}{P(\lambda)}$$

$$= \frac{P(O_1|q_1 = S_i, \lambda) * P(q_1 = S_i, \lambda)}{P(\lambda)}$$

$$= \frac{P(O_1|q_1 = S_i, \lambda) * P(q_1 = S_i|, \lambda) * P(\lambda)}{P(\lambda)}$$

$$= P(O_1|q_1 = S_i, \lambda) * P(q_1 = S_i|, \lambda)$$

$$= \pi_i * b_i(O_1)$$

A su vez, veamos que sucede con $\alpha_t(i)$, donde $t > 1$.

$$\alpha_t(i) = P(O_1, \dots, O_t, q_t = S_i|\lambda)$$

$$= \frac{P(O_1, \dots, O_t, q_t = S_i, \lambda)}{P(\lambda)}$$

$$= \frac{P(O_t|O_1, \dots, O_{t-1}, q_t = S_i, \lambda) * P(O_1, \dots, O_{t-1}, q_t = S_i, \lambda)}{P(\lambda)}$$

En este punto como algún estado debe ocurrir en el momento $t-1$, entonces se cumple que $\sum_{j=1}^{N} P\left(q_{t-1} = S_j\right) = 1$, y por tanto es cierto que

$$\{O_1, \dots, O_{t-1}, q_t = S_i, \lambda\} = \bigcup_{j=1}^{N} \{O_1, \dots, O_{t-1}, q_t = S_i, \boldsymbol{q_{t-1} = S_j}, \lambda\}$$ [10]

Además como los eventos $\{q_{t-1} = S_j\}$ son mutuamente excluyentes para $1 \leq j \leq N$, pues solo un estado puede ocurrir en el momento $t-1$, entonces también se cumple que

$$P(\bigcup_{j=1}^{N} \{O_1, \dots, O_{t-1}, q_t = S_i, q_{t-1} = S_j, \lambda\}) = \sum_{j=1}^{N} P(O_1, \dots, O_{t-1}, q_t = S_i, q_{t-1} = S_j, \lambda)$$ [11]

Continuando con el análisis de $\alpha_t(i)$ tendríamos

$$= \frac{P(O_t|O_1, \dots, O_{t-1}, q_t = S_i, \lambda) * \sum_{j=1}^{N} P(O_1, \dots, O_{t-1}, q_t = S_i, q_{t-1} = S_j, \lambda)}{P(\lambda)}$$

$$= \frac{P(O_t|O_1, \dots, O_{t-1}, q_t = S_i, \lambda) * \sum_{j=1}^{N} [P\left(q_t = S_i|q_{t-1} = S_j, O_1, \dots, O_{t-1}, \lambda\right) * P\left(q_{t-1} = S_j, O_1, \dots, O_{t-1}, \lambda\right)]}{P(\lambda)}$$

[10] Obsérvese que estamos "desmarginalizando" la probabilidad.

[11] Por el tercer axioma de Kolmogorov para la teoría de la probabilidad.

Por (1) y (2), los primeros términos del numerador se pueden simplificar y al continuar aplicando la definición de probabilidad condicional se tiene

$$= \frac{P(O_t|q_t = S_i, \lambda) * \sum_{j=1}^{N}[P(q_t = S_i|q_{t-1} = S_j, \lambda) * P(q_{t-1} = S_j, O_1, \dots, O_{t-1}|\lambda) * P(\lambda)]}{P(\lambda)}$$

Como $P(\lambda)$ no depende de j, entonces

$$= \frac{P(O_t|q_t = S_i, \lambda) * P(\lambda) * \sum_{j=1}^{N}[P(q_t = S_i|q_{t-1} = S_j, \lambda) * P(q_{t-1} = S_j, O_1, \dots, O_{t-1}|\lambda)]}{P(\lambda)}$$

$$= P(O_t|q_t = S_i, \lambda) * \sum_{j=1}^{N}[P(q_t = S_i|q_{t-1} = S_j, \lambda) * P(q_{t-1} = S_j, O_1, \dots, O_{t-1}|\lambda)]$$

$$= b_i(O_t) * \sum_{j=1}^{N}[a_{ji} * \alpha_{t-1}(j)]$$

Este resultado insinúa el procedimiento a seguir para realizar la computación de los $\alpha_T(i)$, para todos los $1 \leq i \leq N$. Ya que cada variable *forward* en tiempo t depende de todas aquellas en tiempo $t-1$. La Fig. 9 ilustra cómo es el esquema de cálculo que detallaremos a continuación.

1) Inicialización (para $1 \leq i \leq N$):

   - $\alpha_1(i) = \pi_i * b_i(O_1)$

2) Recursión (para $2 \leq t \leq T$ y $1 \leq i \leq N$):

   - $\alpha_t(i) = \sum_{j=1}^{N}(\alpha_{t-1}(j) * a_{ji}) * b_i(O_t)$

3) Paso final:

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

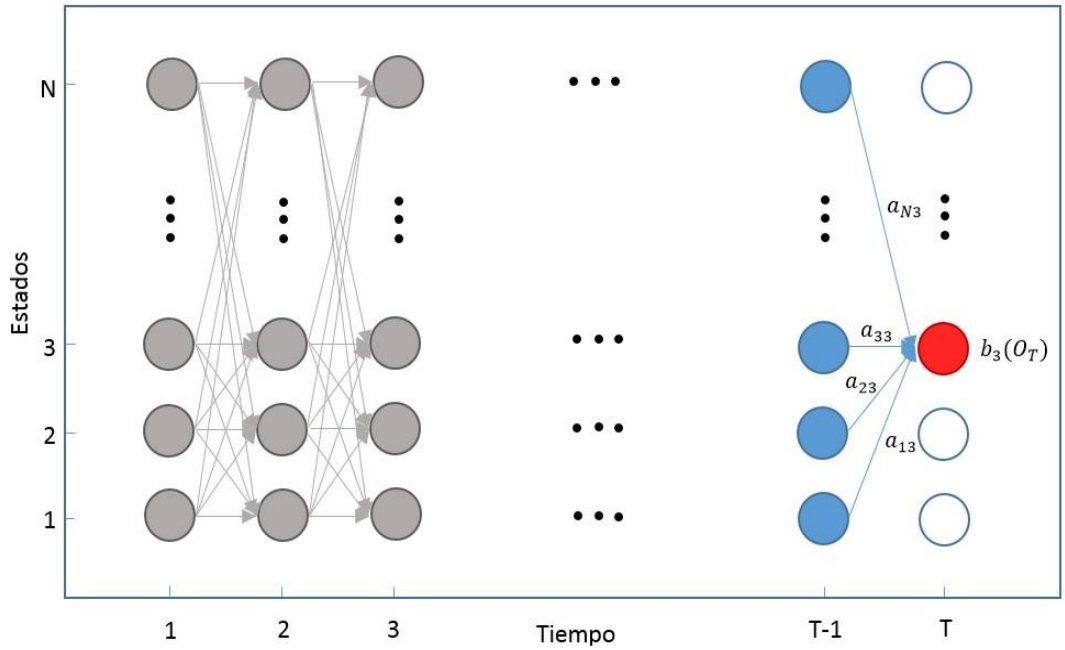

Fig. 9 Muestra la secuencia de operaciones requeridas para el cálculo de la variable *forward* $\alpha_T(3)$ e ilustra cómo se puede aplicar programación dinámica para su cómputo eficiente.

A diferencia de la aproximación inicial para resolver el problema 1 el procedimiento dispuesto en (6) requiere que por cada tiempo t se realice el cálculo de los $\alpha_t(i)$ para todo $1 \leq i \leq N$, lo cual conlleva la suma de N productos para cada uno de ellos. Luego el número de operaciones es TN^2 aproximadamente. Mucho más eficiente que las $2TN^T$ en la solución inicial.

Solución al problema 2

Como se mencionó anteriormente, el problema 2 consiste en elegir cuál es la secuencia de estados que mejor explica a una secuencia de observaciones dada. A diferencia del problema 1 la respuesta aquí no es única, pues depende del criterio que utilicemos, y este a su vez depende del contexto en que nos encontremos. Por ejemplo, un criterio válido puede ser aquel donde elegimos la secuencia de estados tal que $a_{q_1q_2} * a_{q_2q_3} * \dots * a_{q_{T-1}q_T}$ sea máxima; o tal vez podríamos elegir la cadena formada por los estados más probables en cada momento t; o sino aquella donde a cada momento se le asigna el estado que mayores probabilidades tiene de emitir la observación correspondiente acorde a la secuencia dada.

Es obvio que las ventajas y desventajas de cada uno de estos criterios cambian acorde al problema que estemos tratando y que por tanto queda a opinión del desarrollador elegir cual se adapta mejor a sus necesidades. No obstante, se puede observar que estos criterios poseen deficiencias que en la mayor parte de los escenarios los haría inadmisibles. Por ejemplo, el primer criterio que mencionamos no tiene en cuenta las observaciones dadas, y por tanto siempre será el mismo para todas las secuencias de observaciones de longitud similar; el segundo no tiene en cuenta las transiciones entre los estados lo cual puede provocar situaciones donde la secuencia de los estados sea muy poco probable o imposible; mientras que el tercer criterio es similar al primero, pero en vez de obviar a las observaciones este ignora el papel de las transiciones, generando situaciones similares al criterio anterior.

Además de los criterios mencionados existe el *single best state sequence* cuya idea esencial es la de encontrar la secuencia de estados que maximice la probabilidad $P(Q|O,\lambda)$. Veamos esta idea

$$P(Q|O,\lambda) = \frac{P(Q,O,\lambda)}{P(O,\lambda)} = \frac{P(Q,O|\lambda) * P(\lambda)}{P(O|\lambda) * P(\lambda)} = \frac{P(Q,O|\lambda)}{P(O|\lambda)}$$

Luego si el objetivo es encontrar $argmax_{Q\in K}(P(Q|O,\lambda))$, donde K es el conjunto de todas las secuencias de estados de longitud igual al de la secuencia de observaciones (sea esta T), entonces es igual que resolvamos $argmax_{Q\in K}(P(Q,O|\lambda))$ ya que como muestra el resultado anterior $P(Q|O,\lambda) \propto P(Q,O|\lambda)$[12], pues $P(O|\lambda)$ es constante para cualquiera sea la secuencia de estados.

El **Algoritmo de Viterbi** [18] [19] se utiliza para encontrar esta secuencia, y al igual que el algoritmo para la variable *forward*, se basa en métodos de programación dinámica para realizarlos. Consideremos la variable

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

La cual denota la mayor probabilidad de que se genere la secuencia de observaciones hasta un momento t de entre todas las posibles secuencias de estados con igual longitud, pero que su

[12] Este símbolo denota proporcionalidad directa.

último estado (el del momento t) sea S_i. Al igual que con el cálculo de la variable *forward* esta nueva variable también es de carácter recursivo.

Consideremos el caso de $t = 1$. Entonces $\delta_1(i) = P(q_1 = S_1, O_1|\lambda)$, pues no existe secuencia previa, de donde

$$\delta_1(i) = \frac{P(O_1, q_1 = S_i, \lambda)}{P(\lambda)}$$

$$= \frac{P(O_1|q_1 = S_i, \lambda) * P(q_1 = S_i, \lambda)}{P(\lambda)}$$

$$= \frac{P(O_1|q_1 = S_i, \lambda) * P(q_1 = S_i|\lambda) * P(\lambda)}{P(\lambda)}$$

$$= P(O_1|q_1 = S_i, \lambda) * P(q_1 = S_i|\lambda)$$

$$= \pi_i * b_i(O_1)$$

Para $t > 1$, entonces

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

$$= \max_{q_1, q_2, \dots, q_{t-1}} \left(\frac{P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots O_t, \lambda)}{P(\lambda)}\right)$$

$$= \max_{q_1, q_2, \dots, q_{t-1}} \left(\frac{P(O_t|q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots O_{t-1}, \lambda) * P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots O_{t-1}, \lambda)}{P(\lambda)}\right)$$

$$= \max_{q_1, q_2, \dots, q_{t-1}} \left(\frac{P(O_t|q_t = S_i, \lambda) * P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots O_{t-1}, \lambda)}{P(\lambda)}\right)$$

$$= \frac{\max_{q_1, q_2, \dots, q_{t-1}} (P(O_t|q_t = S_i, \lambda) * P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots O_{t-1}, \lambda))}{P(\lambda)}$$

Nótese que $P(\lambda)$ es constante sin importar la secuencia $q_1, q_2, \dots, q_{t-1}$.

$$= \frac{P(O_t|q_t = S_i, \lambda) * \max_{q_1, q_2, \dots, q_{t-1}} (P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots O_{t-1}, \lambda))}{P(\lambda)}$$

Por (2) se cumple que $P(O_t|q_t = S_i, \lambda)$ es independiente de cualquier secuencia de estados

$$= b_i(O_t) * \frac{\max_{q_{t-1} = S_j} \left(\max_{q_1, q_2, \dots, q_{t-2}} (P(q_1, q_2, \dots, q_{t-1} = S_j, q_t = S_i, O_1, O_2, \dots O_{t-1}, \lambda))\right)}{P(\lambda)}$$

Obsérvese que lo único que se ha hecho es denotar como S_j al estado que se halla en el momento $t - 1$ de la secuencia con la máxima probabilidad y separar ambos máximos.

$$= b_i(O_t) * \frac{\max_{q_{t-1} = S_j} \left(\max_{q_1, q_2, \dots, q_{t-2}} (P(q_t = S_i|q_{t-1} = S_j) * P(q_1, q_2, \dots, q_{t-1} = S_j, O_1, O_2, \dots O_{t-1}, \lambda))\right)}{P(\lambda)}$$

$$= b_i(O_t) * \frac{\max_{q_{t-1} = S_j} (a_{ji} * \max_{q_1, q_2, \dots, q_{t-2}} (P(q_1, q_2, \dots, q_{t-1} = S_j, O_1, O_2, \dots O_{t-1}, \lambda)))}{P(\lambda)}$$

$$= b_i(O_t) * \frac{\max\limits_{q_{t-1}=S_j}(a_{ji} * \max\limits_{q_1,q_2,\dots,q_{t-2}}(P(q_1,q_2,\dots,q_{t-1}=S_j,O_1,O_2,\dots O_{t-1}|\lambda) * P(\lambda)))}{P(\lambda)}$$

$$= b_i(O_t) * \max_{q_{t-1}=S_j}(a_{ji} * \max_{q_1,q_2,\dots,q_{t-2}}(P(q_1,q_2,\dots,q_{t-1}=S_j,O_1,O_2,\dots O_{t-1}|\lambda))$$

$$= b_i(O_t) * \max_{q_{t-1}=S_j}(a_{ji} * \delta_{t-1}(j))$$

Intuitivamente las variables $\delta_{t-1}(j)$ contienen el valor de la probabilidad más alta de entre todos los caminos de longitud $t-1$ que terminen en el estado S_j. Luego, para saber $\delta_t(i)$, hay que elegir de entre todos los estados del momento anterior cuáles realizan la mejor transición hacia S_i (esto es $\max\limits_{q_{t-1}=S_j}(a_{ji} * \delta_{t-1}(j))$) para entonces añadirle la probabilidad de la observación correspondiente.

Sin embargo, las variables $\delta_t(i)$ no permiten por sí solas recuperar la secuencia de estados cuya probabilidad representan. Por tanto el procedimiento completo también incluye a la variable $\psi_t(i) = \operatorname{argmax}\limits_{1\le j\le N}(a_{ji} * \delta_{t-1}(j))$, la cual denota para un estado i en el momento t cuál es el estado que más probabilidad tiene de ser su antecesor. Veamos el procedimiento completo.

1) Inicialización (para $1 \le i \le N$):

   - $\delta_1(i) = \pi_i * b_i(O_1)$

2) Recursión (para $2 \le t \le T$ y $1 \le i \le N$):

   - $\delta_t(i) = \max\limits_{q_{t-1}=S_j}\left(a_{ji} * \delta_{t-1}(q_{t-1})\right) * b_i(O_t)$
   - $\psi_t(i) = \operatorname{argmax}\limits_{1\le j\le N}(a_{ji} * \delta_{t-1}(j))$

3) Paso final:

   - $scoreBestPath = \max\limits_{1\le i\le N}(\delta_T(i))$
   - ${q^*}_T = \operatorname{argmax}\limits_{1\le i\le N}(\delta_T(i))$

4) Obtener mejor camino (para $t = T-1, T-2, \dots, 1$):

   - ${q^*}_t = \psi_{t+1}({q^*}_{t+1})$

Donde los ${q^*}_t$ con $1 \leq t \leq T$ representan los estados del mejor camino. La Fig. 10 muestra un diagrama que ejemplifica este procedimiento.

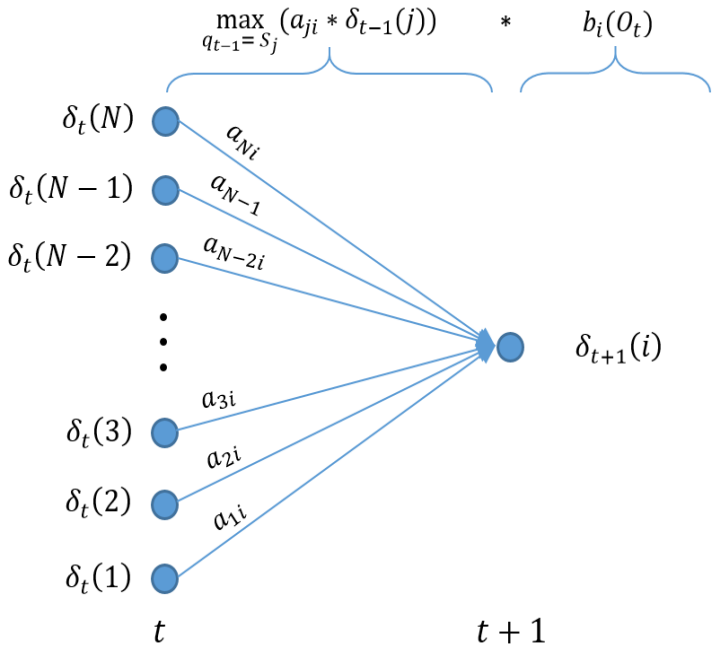

Fig.10 Diagrama de cómo realizar el cómputo de las variables $\delta_t(i)$.

El algoritmo de Viterbi recuerda al descrito anteriormente para el cálculo de las variables *forward*, las únicas diferencias son el uso de la función max y el *backtracking* al final del procedimiento. Analizando el número de operaciones computacionales que se deben realizar para este algoritmo observamos que por cada momento t se realiza el cálculo de la variable $\delta_t(i)$ para cada uno de los posibles estados, lo cual requiere la elección del mayor producto entre los $\delta_{t-1}(j)$ previos y la probabilidad de transición al estado i correspondiente. A su vez el cómputo de la variable $\psi_t(i)$ no requiere ningún trabajo extra si es correctamente implementada, ya que su valor puede ser obtenido durante el cálculo de $\delta_t(i)$. Visto esto el número de operaciones asciende a TN^2, al igual que en el problema anterior. Sin embargo, el *backtracking* que se realiza al final para obtener el camino adiciona unas T operaciones extras.

Solución al problema 3

El tercer problema que se ha descrito para los HMM es aquel donde a partir de una secuencia de observaciones $O = O_1, \dots, O_T$ determinamos cuáles son los parámetros del modelo $\lambda = (N, A, B, \pi)$[13] que mejor se adaptan a esta. O sea aquellos que maximizan el valor de $P(O|\lambda)$.

No obstante, este problema no posee solución óptima. Si considerásemos la función $f_O: \Lambda \rightarrow [0,1]$ cuyo dominio Λ es el conjunto de todos los posibles modelos de HMMs, mientras que su codominio corresponde a la probabilidad de que un modelo de Λ genere a O. Luego, encontrar los valores óptimos de un modelo λ para una observación O es lo mismo que encontrar el máximo global de la función f_O. La Fig. 11 es una representación gráfica de cómo podría lucir esta función.

[13] Obsérvese hemos dejado fuera a M. Esto es porque no consideraremos observaciones discretas y/o finitas.

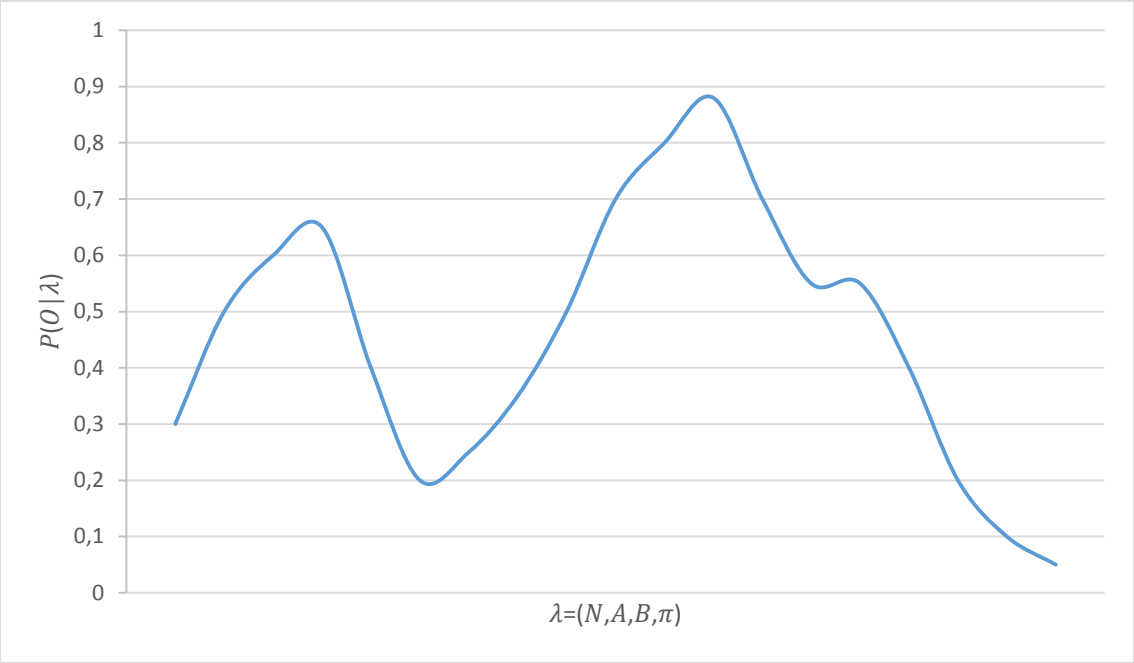

Fig. 11 Ejemplificación de cómo podrían cambiar los valores de $P(O|\lambda)$ dependiendo del modelo λ suministrado.

El **procedimiento iterativo de Baum-Welch** [1] [17], es uno de los algoritmos que se emplea para determinar los parámetros de un modelo dado una secuencia de observaciones. Sin embargo, debido a lo explicado anteriormente, solo se puede asegurar su convergencia a un máximo local de f_O. Para describir este procedimiento volvamos al *forward-backward algorithm* del cual nos faltó especificar como calcular las variables *backward*. Sea

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

La cual denota la probabilidad de que se genere la secuencia de observaciones desde el momento $t+1$ hasta el final sin importar la secuencia de estados que se use, excepto por el que se ubica en el momento t, del cual se sabe que es el estado i.

Al igual que antes las variables *backward* también son recursivas.

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

$$= \frac{P(O_{t+1}, \dots O_T, q_t = S_i, \lambda)}{P(q_t = S_i, \lambda)}$$

$$= \frac{\sum_{j=1}^{N} P(O_{t+1}, \dots O_T, q_t = S_i, q_{t+1} = S_j, \lambda)}{P(q_t = S_i, \lambda)}$$

$$= \frac{\sum_{j=1}^{N} P\left(O_{t+1} \middle| q_{t+1} = S_j, O_{t+2}, \dots O_T, q_t = S_i, \lambda\right) * P(O_{t+2}, \dots O_T, q_t = S_i, q_{t+1} = S_j, \lambda)}{P(q_t = S_i, \lambda)}$$

$$= \frac{\sum_{j=1}^{N} P\left(O_{t+1} \middle| q_{t+1} = S_j, \lambda\right) * P\left(O_{t+2}, \dots O_T \middle| q_t = S_i, q_{t+1} = S_j, \lambda\right) * P(q_t = S_i, q_{t+1} = S_j, \lambda)}{P(q_t = S_i, \lambda)}$$

$$= \frac{\sum_{j=1}^{N} b_j(O_{t+1}) * P\left(O_{t+2}, \dots O_T \middle| q_t = S_i, q_{t+1} = S_j, \lambda\right) * P\left(q_{t+1} = S_j \middle| q_t = S_i, \lambda\right) * P(q_t = S_i, \lambda)}{P(q_t = S_i, \lambda)}$$

$= \sum_{j=1}^{N} a_{ij} * b_j(O_{t+1}) * P(O_{t+2}, \dots O_T | q_t = S_i, q_{t+1} = S_j, \lambda)$

$= \sum_{j=1}^{N} a_{ij} * b_j(O_{t+1}) * P(O_{t+2}, \dots O_T | q_{t+1} = S_j, \lambda)$

En este paso hemos usado $O_{t+2:T} \perp q_t = S_i | q_{t+1} = S_j$. Lo cual se justifica gracias a (1) que nos asegura que tanto los estados como las observaciones futuras son independientes del pasado conocido el presente. En este caso, el presente sería $t + 1$. No obstante, este es un resultado intuitivo si se parte del hecho más simple que los estados futuros son independientes de los pasados dado el actual, esto combinado a que las observaciones son independientes a todo dado el estado que las genera, entonces es imposible que los estados pasados puedan afectar las observaciones futuras conocido el estado actual, recuérdese la Fig. 8. Al final se cumple que

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} * b_j(O_{t+1}) * \beta_{t+1}(j)$$

En cuanto al valor de $\beta_T(i)$ para $1 \leq i \leq N$, observemos que $\beta_T(i) = P(\phi | q_T = S_i, \lambda) = 1$, pues el conjunto vacío pertenece a todo conjunto. A continuación se desglosa el procedimiento a seguir para el cálculo de los $\beta_t(i)$, el cual además se ilustra en la Fig. 12.

1) Inicialización (para $1 \leq i \leq N$):

   - $\beta_T(i) = 1$

2) Recursión (para $1 \leq i \leq N$ y $t = T - 1, T - 2, \dots, 1$):

   - $\beta_t(i) = \sum_{j=1}^{N} a_{ij} * b_j(O_{t+1}) * \beta_{t+1}(j)$

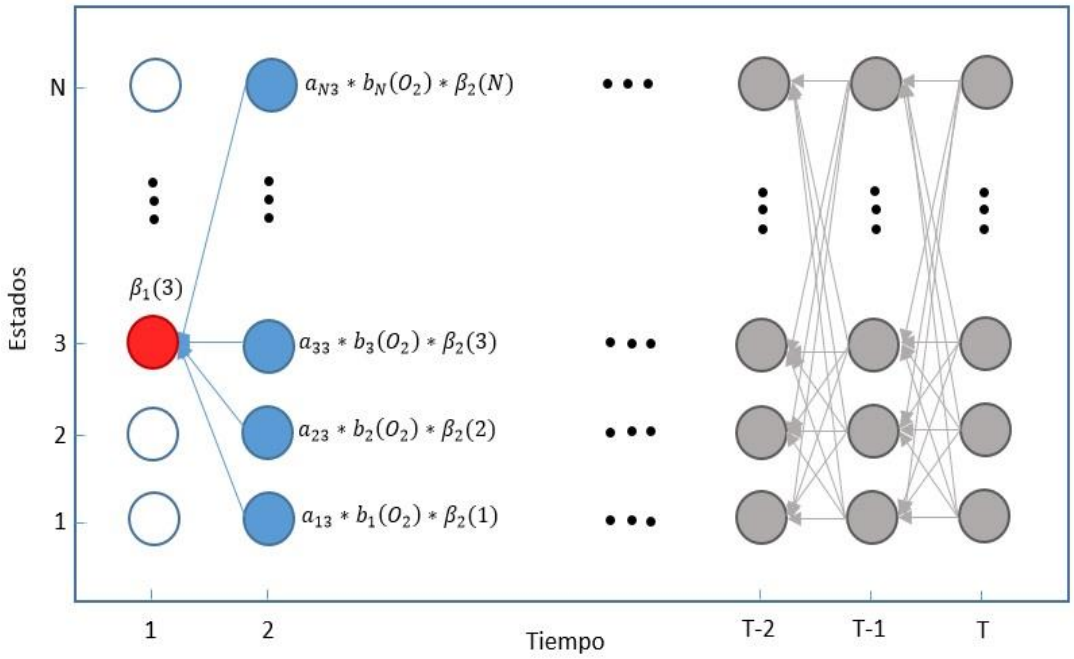

Fig. 12 Muestra la secuencia de operaciones requeridas para el cálculo de la variable *backward* $\beta_1(3)$ e ilustra cómo se puede aplicar programación dinámica para su cómputo eficiente.

Ahora que ya hemos visto cómo realizar el *forward-backward algorithm* volvamos al asunto en cuestión, cómo encontrar un modelo λ adecuado que maximice (al menos localmente) $P(O|\lambda)$. Para esto definamos la variable

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | O, \lambda) \tag{7}$$

La cual denota la probabilidad de que ocurra una transición del estado i al j en el momento t, conocidas la secuencia de observaciones y el modelo. Veamos cómo se calcula esta variable, y luego como se utiliza para estimar los parámetros del modelo.

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

$$= \frac{P(q_t = S_i, q_{t+1} = S_j, O, \lambda)}{P(O, \lambda)}$$

$$= \frac{P(q_{t+1} = S_j, q_t = S_i, O_1, \dots, O_t, O_{t+1}, \dots, O_T, \lambda)}{P(O, \lambda)}$$

$$= \frac{P\left(O_{t+1} | q_{t+1} = S_j, q_t = S_i, O_1, \dots, O_t, O_{t+2} \dots, O_T, \lambda\right) * P(q_{t+1} = S_j, q_t = S_i, O_1, \dots, O_t, O_{t+2} \dots, O_T, \lambda)}{P(O, \lambda)}$$

$$= \frac{b_j(O_{t+1}) * P\left(O_{t+2} \dots, O_T | q_{t+1} = S_j, q_t = S_i, O_1, \dots, O_t, \lambda\right) * P(q_{t+1} = S_j, q_t = S_i, O_1, \dots, O_t, \lambda)}{P(O, \lambda)}$$

$$= \frac{b_j(O_{t+1}) * \beta_{t+1}(j) * P\left(q_{t+1} = S_j | q_t = S_i, O_1, \dots, O_t, \lambda\right) * P(q_t = S_i, O_1, \dots, O_t, \lambda)}{P(O, \lambda)}$$

$$= \frac{b_j(O_{t+1}) * \beta_{t+1}(j) * P\left(q_{t+1} = S_j | q_t = S_i, \lambda\right) * P(q_t = S_i, O_1, \dots, O_t | \lambda) * P(\lambda)}{P(O|\lambda) * P(\lambda)}$$

$$= \frac{\alpha_t(i) * a_{ij} * b_j(O_{t+1}) * \beta_{t+1}(j)}{\sum_{i=1}^{N} \alpha_T(i)}$$

El numerador de esta expresión se puede interpretar como la probabilidad de todos los caminos posibles, siempre y cuando cumplan con las observaciones y que pasen por el estado i y el j en los momentos t y $t+1$ respectivamente. Mientras que el denominador es la probabilidad de todos los caminos posibles que cumplan con las observaciones, lo cual le permite normalizar los valores de $\xi_t(i,j)$ para todo par $1 \leq i, j \leq N$. La Fig. 13 muestra este procedimiento.

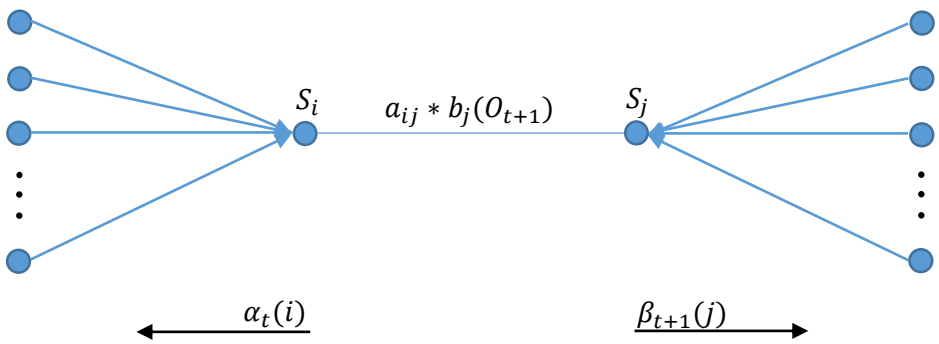

Fig. 13 Ilustración del significado de la probabilidad representada por la variable $\xi_t(i,j)$.

A partir de la variable $\xi_t(i,j)$ se puede definir otra de gran importancia para el problema en cuestión, esta es $\gamma_t(i) = P(q_t = S_i|O,\lambda)$ la cual representa la probabilidad de que en el momento t el estado i ocurra dado el modelo y la secuencia de observaciones. Luego

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{8}$$

Si para ambas variables en (7) y (8) sumamos su valor para todos los valores posibles de t (aunque en el caso de (7) solo se hará hasta $T-1$), entonces obtenemos una cantidad que puede ser interpretada como el número esperado de transiciones de un estado i a otro j y como el número esperado de veces en que ocurre el estado i, respectivamente. A partir de aquí se pueden formular las siguientes expresiones para el cálculo de los parámetros A y π del modelo $\lambda = (N, A, B, \pi)$[14].

$$\bar{\pi}_i = \gamma_1(i) \tag{9}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{10}$$

La expresión (9) es muy simple ya que solo denota la probabilidad de inicio para cada estado, mediante el número esperado de veces que se espera ocurra este estado en el momento inicial. La expresión (10) utiliza, como se esperaría, el número esperado de transiciones de i a j, pero emplea el número esperado de ocurrencias del estado i para normalizar este valor y poder garantizar las restricciones necesarias para que las entradas de A sean probabilidades.

El procedimiento iterativo de Baum-Welch utiliza estos resultados para dar respuesta al problema 3 y consiste en utilizar el modelo inicial $\lambda = (A, B, \pi)$ en el cómputo de (9) y (10) para luego definir un nuevo modelo $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$ a partir de los valores calculados. Ha sido probado en [1] y [17] que $P\left(O\middle|\bar{\lambda}\right) \geq P(O|\lambda)$.

Observaciones de densidad continua

Como se mencionó antes las observaciones representan a la señal que emiten los procesos naturales por lo que solo a través de ellas es que podemos estudiarlos. Estas observaciones pueden ser discretas o continuas, dependiendo del proceso. Sin embargo, debido al predominio de los procesos analógicos en la naturaleza estos últimas son las más comunes, por lo que se hace necesario encontrar una forma adecuada de representar las probabilidades de emisión de estas observaciones (esto es B) que vaya más allá de una simple función de distribución discreta y que recoja la probabilidad de cada una de las posibles observaciones para cada estado.

El reto principal de usar una función de densidad (*pdf*, por sus siglas en inglés) para la definición de B es encontrar para cuales de ellas existe un método de reestimación eficiente que pueda ser añadido al procedimiento de Baum-Welch que se explicó en la sección anterior.

Afortunadamente existe una representación general de una *pdf* para la cual se ha formulado un procedimiento de reestimación [20] [21], esta consiste en una mezcla finita de la siguiente forma

$$b_i(O) = \sum_{m=1}^{M} c_{im} * \Re(O, \mu_{jm}, U_{jm}), \quad 1 \leq i \leq N \tag{11}$$

Donde O[15] es la observación (por lo general un vector) de la cual queremos saber su probabilidad de emisión en el estado j; M es el número de mezclas; c_{im} es el coeficiente

[14] Más adelante veremos cómo calcular el resto de los parámetros.

[15] Nótese que hasta ahora hemos utilizado O como representación de una secuencia de observaciones. Esto es diferente del uso actual donde representa una específica.

correspondiente a la mezcla m-ésima en el estado j; $\Re$ es cualquier función de densidad elípticamente simétrica [20] (por ejemplo la distribución normal); μ_{jm} es la media y U_{jm} es la matriz de covarianza para la m-ésima función $\Re$. También se debe agregar que los coeficientes de las mezclas cumplen las restricciones estocásticas, esto es

$$c_{im} \geq 0, \quad 1 \leq m \leq M, 1 \leq i \leq N$$

$$\sum_{m=1}^{M} c_{im} = 1, \quad 1 \leq i \leq N$$

Lo cual asegura que la *pdf* esté correctamente normalizada [21]

$$\int_{-\infty}^{\infty} b_i(x)\, dx = 1, \quad 1 \leq i \leq N$$

Ha sido demostrado en [20] que las funciones de reestimación para los valores citados antes son:

$$\bar{c}_{im} = \frac{\sum_{t=1}^{T} \gamma_t(i,m)}{\sum_{t=1}^{T} \sum_{m=1}^{M} \gamma_t(i,m)}$$

$$\bar{\mu}_{im} = \frac{\sum_{t=1}^{T} \gamma_t(i,m) * O_t}{\sum_{t=1}^{T} \gamma_t(i,m)}$$

$$\bar{U}_{im} = \frac{\sum_{t=1}^{T} \gamma_t(i,m) * (O_t - \mu_{im})(O_t - \mu_{im})'}{\sum_{t=1}^{T} \gamma_t(i,m)}$$

Donde $\gamma_t(i,m)$ representa la probabilidad de estar en el estado i en el momento t usando la m-ésima mezcla para emitir a O_t.

$$\gamma_t(i,m) = \left[\frac{\alpha_t(i) * \beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i) * \beta_t(i)}\right] * \left[\frac{c_{im} * \Re(O_t, \mu_{jm}, U_{jm})}{\sum_{m=1}^{M} c_{im} * \Re(O_t, \mu_{jm}, U_{jm})}\right]$$

No veremos la demostración de cada una de las fórmulas de reestimación anteriores debido a que su complejidad supera las intenciones de este trabajo, sin embargo [20] y [21] muestran este trabajo.

Escalamiento

Si se observa el procedimiento para obtener las variables *forward*, entonces notaremos que todo $\alpha_t(i)$ se compone de la siguiente manera

$$\alpha_t(i) = b_i(O_t) * \sum_{j=1}^{N} (\alpha_{t-1}(j) * a_{ji}) \tag{12}$$

Luego, a medida que t incrementa, los valores de estas probabilidades se van haciendo cada vez más pequeños debido al producto consecutivo de términos entre 0 y 1. Con el objetivo de contrarrestar este comportamiento que rápidamente puede superar el rango de precisión de la computadora, cada valor de $\alpha_t(i)$ es escalado, y a su vez, este nuevo valor es utilizado en el cálculo de los próximos $\alpha_{t+1}(i)$.

Debido a que el decrecimiento de las *forward* se debe esencialmente al aumento de t, se ha determinado que el valor asignado al factor de escala debe depender del tiempo, más que de los estados, razón por la cual el factor de escala c_t es independiente de i.

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)}$$

Luego si definimos a $\alpha''_t(i) = c_t * \alpha'_t(i)$ como el valor escalado de $\alpha'_t(i)$, entonces $\alpha'_t(i) = b_i(O_t) * \sum_{j=1}^{N}(\alpha''_{t-1}(j) * a_{ji})$.

Demostremos ahora que debido al procedimiento anterior la relación que existe entre los $\alpha''_t(i)$ y los $\alpha_t(i)$ reales es

$$\alpha''_t(i) = \prod_{k=1}^{t} c_k * \alpha_t(i)$$

Para probarlo realicemos una inducción sobre t. Si $t = 1$, entonces $\alpha'_1(i) = \pi_i * b_i(O_1) = \alpha_1(i)$, por lo que $\alpha''_1(i) = c_1 * \alpha'_1(i) = \prod_{k=1}^{1} c_k * \alpha_1(i)$.

Supongamos que para $t = n$ se cumple que $\alpha''_n(i) = \prod_{k=1}^{n} c_k * \alpha_n(i)$.

Sea $t = n + 1$, entonces $\alpha'_{n+1}(i) = b_i(O_{n+1}) * \sum_{j=1}^{N}(\alpha''_n(j) * a_{ji})$. Por la hipótesis de inducción $\alpha'_{n+1}(i) = b_i(O_{n+1}) * \sum_{j=1}^{N}(\prod_{k=1}^{n}(c_k * \alpha_n(j)) * a_{ji})$. Como

$\alpha''_{n+1}(i) = c_{n+1} * \alpha'_{n+1}(i)$

$= c_{n+1} * b_i(O_{n+1}) * \sum_{j=1}^{N}(\prod_{k=1}^{n}(c_k * \alpha_n(j)) * a_{ji})$

$= c_{n+1} * \prod_{k=1}^{n} c_k * b_i(O_{n+1}) * \sum_{j=1}^{N}(\alpha_n(j) * a_{ji})$

$= \prod_{k=1}^{n+1} c_k * \alpha_{n+1}(i)$

Luego por el principio de inducción matemática se cumple que $\forall t \in \mathbb{N}$ se cumple que

$$\alpha''_t(i) = \prod_{k=1}^{t} c_k * \alpha_t(i)$$

Las variables *backward* presentan también el problema de superar el rango dinámico de la computadora, por lo que se hace necesario realizarles este proceso de escalamiento. Sin embargo, en vez de usar el factor de escala intuitivo $\sum_{i=1}^{N} \beta_t(i)$ se usará, por razones de cómputo, y otras que veremos más adelante el mismo factor c_t. Por lo que

$$\beta''_t(i) = \prod_{k=t}^{N} c_k * \beta_t(i)$$

Luego si usamos estos valores escalados en las expresiones (9) y (10) veremos que estos no afectan el resultado de las expresiones de reestimación

$$\bar{\pi}_i = \gamma_1(i) = \sum_{j=1}^{N} \xi_1(i,j) = \sum_{j=1}^{N} \frac{\alpha''_1(i) * a_{ij} * b_j(O_1) * \beta''_2(j)}{\sum_{k=1}^{N} \alpha''_T(k)}$$

$$= \sum_{j=1}^{N} \frac{c_1 * \alpha_1(i) * a_{ij} * b_j(O_1) * \prod_{l=2}^{T} c_l * \beta_2(j)}{\sum_{k=1}^{N} \prod_{l=1}^{T} c_l * \alpha_T(k)}$$

$$= \sum_{j=1}^{N} \frac{\prod_{l=1}^{T} c_l * \alpha_1(i) * a_{ij} * b_j(O_1) * \beta_2(j)}{\prod_{l=1}^{T} c_l * \sum_{k=1}^{N} \alpha_T(k)}$$

$$= \sum_{j=1}^{N} \frac{\alpha_1(i) * a_{ij} * b_j(O_1) * \beta_2(j)}{\sum_{k=1}^{N} \alpha_T(k)}$$

Mientras que para (10)

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)} = \frac{\sum_{t=1}^{T} (\alpha''_t(i) * a_{ij} * b_j(O_t) * \beta''_{t+1}(j))}{\sum_{k=1}^{N} \sum_{t=1}^{T} (\alpha''_t(i) * a_{ik} * b_k(O_t) * \beta''_{t+1}(k))}$$

$$= \frac{\sum_{t=1}^{T} (\prod_{l=1}^{t} c_l * \alpha_t(i) * a_{ij} * b_j(O_t) * \prod_{l=t+1}^{T} c_l * \beta_{t+1}(j))}{\sum_{k=1}^{N} \sum_{t=1}^{T} (\prod_{l=1}^{t} c_l * \alpha_t(i) * a_{ik} * b_k(O_t) * \prod_{l=t+1}^{T} c_l * \beta_{t+1}(k))}$$

$$= \frac{\sum_{t=1}^{T} (\prod_{l=1}^{T} c_l * \alpha_t(i) * a_{ij} * b_j(O_t) * \beta_{t+1}(j))}{\sum_{k=1}^{N} \sum_{t=1}^{T} (\prod_{l=1}^{T} c_l * \alpha_t(i) * a_{ik} * b_k(O_t) * \beta_{t+1}(k))}$$

$$= \frac{\prod_{l=1}^{T} c_l * \sum_{t=1}^{T} (\alpha_t(i) * a_{ij} * b_j(O_t) * \beta_{t+1}(j))}{\prod_{l=1}^{T} c_l * \sum_{k=1}^{N} \sum_{t=1}^{T} (\alpha_t(i) * a_{ik} * b_k(O_t) * \beta_{t+1}(k))}$$

$$= \frac{\sum_{t=1}^{T} (\alpha_t(i) * a_{ij} * b_j(O_t) * \beta_{t+1}(j))}{\sum_{k=1}^{N} \sum_{t=1}^{T} (\alpha_t(i) * a_{ik} * b_k(O_t) * \beta_{t+1}(k))}$$

En el caso de las funciones de reestimación para el caso de las observaciones con densidad continua $\bar{c}_{im}$, $\bar{\mu}_{im}$ y $\overline{U}_{im}$ la presencia de las variables *forward* y *backward* se encuentran en los términos $\gamma_t(i,m)$. Veamos si los valores escalados de estas afectan la fórmula de reestimación de estos términos.

$$\gamma''_t(i,m) = \left[\frac{\alpha''_t(i) * \beta''_t(i)}{\sum_{i=1}^{N} \alpha''_t(i) * \beta''_t(i)}\right] * \left[\frac{c_{im} * \Re(O_t, \mu_{jm}, U_{jm})}{\sum_{m=1}^{M} c_{im} * \Re(O, \mu_{jm}, U_{jm})}\right]$$

$$= \left[\frac{\prod_{l=1}^{t} c_l * \alpha_t(i) * \prod_{l=t}^{T} c_l * \beta_{t+1}(j)}{\sum_{i=1}^{N} \prod_{l=1}^{t} c_l * \alpha_t(i) * \prod_{l=t}^{T} c_l * \beta_{t+1}(j)}\right] * \left[\frac{c_{im} * \Re(O_t, \mu_{jm}, U_{jm})}{\sum_{m=1}^{M} c_{im} * \Re(O, \mu_{jm}, U_{jm})}\right]$$

$$= \left[\frac{\prod_{l=1}^{T} c_l * c_t * \alpha_t(i) * \beta_t(i)}{\prod_{l=1}^{T} c_l * c_t \sum_{i=1}^{N} \alpha_t(i) * \beta_t(i)}\right] * \left[\frac{c_{im} * \Re(O_t, \mu_{jm}, U_{jm})}{\sum_{m=1}^{M} c_{im} * \Re(O, \mu_{jm}, U_{jm})}\right]$$

$$= \left[\frac{\alpha_t(i) * \beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i) * \beta_t(i)}\right] * \left[\frac{c_{im} * \Re(O_t, \mu_{jm}, U_{jm})}{\sum_{m=1}^{M} c_{im} * \Re(O, \mu_{jm}, U_{jm})}\right]$$

$$= \gamma_t(i,m)$$

Por lo que $\gamma_t(i,m)$ no es afectado por el uso de las variables escaladas, así que las funciones de reestimación para los parámetros de las mezclas se mantendrían idénticas. El único cambio real al procedimiento de reestimación es durante el cálculo de $P(O|\lambda)$, pues no se pueden sumar simplemente los $\alpha''_T(i)$ para todos los estados ya que estos se encuentran escalados. No obstante debido a que

$$\sum_{i=1}^{N} \alpha''_T(i) = \sum_{i=1}^{N} \prod_{k=1}^{T} c_k * \alpha_T(i)$$

$= \prod_{k=1}^{T} c_k * \sum_{i=1}^{N} \alpha_T(i)$

$= \prod_{k=1}^{T} c_k * P(O|\lambda)$

Entonces

$$P(O|\lambda) = \frac{\sum_{i=1}^{N} \alpha''_T(i)}{\prod_{k=1}^{T} c_k}$$

$$\log P(O|\lambda) = \log \sum_{i=1}^{N} \alpha''_T(i) - \log \prod_{k=1}^{T} c_k$$

Obsérvese que podemos obtener el $\log P(O|\lambda)$ y no el valor directamente pues $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$ lo cual se encontraría por debajo del rango dinámico de la máquina.

En cuanto al algoritmo de Viterbi cada variable de la forma $\phi_t(i)$ contiene la probabilidad de la secuencia de estados con la mayor posibilidad de generar las observaciones hasta ese instante y concluir en el estado i. Luego para largos caminos (o sea grandes valores de t) esta probabilidad puede hacerse muy pequeña. No obstante, en este caso no es necesario aplicar un mecanismo de escalamiento tan complejo como el anterior ya que aplicando la operación logarítmica durante el proceso se resuelve este problema. Luego si definimos

$$\delta_t(i) = \max_{q_1, q_2 \cdots, q_{t-1}} [\log P(q_1, q_2, \dots, q_t = S_i, O_1, O_2, \dots, O_t|\lambda)]$$

El procedimiento seguiría de la siguiente manera

1) Inicialización (para $1 \leq i \leq N$):

   - $\delta_1(i) = \log \pi_i + \log b_i(O_1)$

2) Recursión (para $2 \leq t \leq T$ y $1 \leq i \leq N$):

   - $\delta_t(i) = \max_{q_{t-1} = S_j} \left(\log a_{ji} + \delta_{t-1}(q_{t-1})\right) + \log b_i(O_t)$
   - $\psi_t(i) = \operatorname{argmax}_{1 \leq j \leq N}(\log a_{ji} + \delta_{t-1}(j))$

3) Paso final:

   - $\log(scoreBestPath) = \max_{1 \leq i \leq N}\left(\delta_T(i)\right)$
   - $q^*_T = \operatorname{argmax}_{1 \leq i \leq N}\left(\delta_T(i)\right)$

4) Obtener mejor camino (para $t = T-1, T-2, \dots, 1$):

   - $q^*_t = \psi_{t+1}(q^*_{t+1})$

Por tanto obtenemos un procedimiento sin los problemas numéricos del anterior, pero que cumple con los mismos objetivos.

Múltiples secuencias de observaciones

El procedimiento que hemos explicado hasta este punto para el entrenamiento de los parámetros del modelo se basa en el uso de una única secuencia de observaciones. Sin embargo, es fácil comprender por qué esta aproximación puede resultar ineficiente en la mayor parte de los experimentos, pues una secuencia de observaciones corresponde a una ocurrencia específica del proceso asociado, lo cual debido a la naturaleza aleatoria de estos sistemas no representaría un exponente fidedigno de las características estadísticas de dicho proceso. Por tanto para lograr este objetivo, mientras más secuencias de observaciones se utilicen para entrenar al modelo [22], entonces más parecido será este al proceso real.

Consideremos $O = [O^{(1)}, O^{(2)}, \dots, O^{(K)}]$ donde $O^{(l)} = [O_1^{(l)}, O_2^{(l)}, \dots, O_{T_l}^{(l)}]$ es la l -ésima secuencia de observaciones. Luego, como las fórmulas de reestimación se basan en las ocurrencias de ciertos eventos para determinar los valores estadísticos del modelo, entonces la extensión de la expresión (10), sería

$$\bar{a}_{ij} = \frac{\sum_{l=1}^{K}(\sum_{t=1}^{T_l-1} {\alpha^l}_t(i) * a_{ij} * b_j(O_{t+1}) * {\beta^l}_{t+1}(j))}{\sum_{l=1}^{K}(\sum_{t=1}^{T_l-1}\sum_{j=1}^{N} {\alpha^l}_t(i) * a_{ij} * b_j({O^l}_{t+1}) * {\beta^l}_{t+1}(j))}$$

O sea aunar las ocurrencias de transiciones del estado i al j para todas las secuencias de observaciones sobre el numero de veces que se encontró en el estado i (nuevamente en todas las secuencias de observaciones). Nótese la diferencia con la expresión inicial que solo consideraba el número de ocurrencias de estos eventos en una única secuencia de observaciones. Es de notar que ${\alpha^l}_t(i)$ se refiere al $\alpha_t(i)$ para la secuencia de observaciones O^l, sucediendo lo mismo para los ${\beta^l}_t(i)$.

Sin embargo, si utilizamos el proceso de escalamiento antes descrito como cada secuencia de observaciones posee un factor de escala diferente, entonces

$$\bar{a}_{ij} = \frac{\sum_{l=1}^{K}(\sum_{t=1}^{T_l-1} {\alpha^{l''}}_t(i) * a_{ij} * b_j(O_{t+1}) * {\beta^{l''}}_{t+1}(j))}{\sum_{l=1}^{K}(\sum_{t=1}^{T_l-1}\sum_{j=1}^{N} {\alpha^{l''}}_t(i) * a_{ij} * b_j({O^l}_{t+1}) * {\beta^{l''}}_{t+1}(j))}$$

$$= \frac{\sum_{l=1}^{K}(\sum_{t=1}^{T_l-1}\prod_{k=1}^{t} {c^l}_k * {\alpha^l}_t(i) * a_{ij} * b_j(O_{t+1}) * \prod_{k=t+1}^{T_l} {c^l}_k * {\beta^l}_{t+1}(j))}{\sum_{l=1}^{K}(\sum_{t=1}^{T_l-1}\sum_{j=1}^{N}\prod_{k=1}^{t} {c^l}_k * {\alpha^l}_t(i) * a_{ij} * b_j({O^l}_{t+1}) * \prod_{k=t+1}^{T_l} {c^l}_k * {\beta^l}_{t+1}(j))}$$

$$= \frac{\sum_{l=1}^{K}\prod_{k=1}^{T_l} {c^l}_k * (\sum_{t=1}^{T_l-1} {\alpha^l}_t(i) * a_{ij} * b_j(O_{t+1}) * {\beta^l}_{t+1}(j))}{\sum_{l=1}^{K}\prod_{k=1}^{T_l} {c^l}_k * (\sum_{t=1}^{T_l-1}\sum_{j=1}^{N} {\alpha^l}_t(i) * a_{ij} * b_j({O^l}_{t+1}) * {\beta^l}_{t+1}(j))}$$

Luego los términos $\prod_{k=1}^{T} {c^l}_k$ no pueden ser simplificados, ya que dependen de la secuencia de observaciones y por tanto, cambiarían el resultado de la función de reestimación. Con el objetivo de deshacernos de estos valores las funciones de reestimación quedarían así

$$\bar{a}_{ij} = \frac{\sum_{l=1}^{K}\frac{1}{\prod_{k=1}^{T_l} {c^l}_k}(\sum_{t=1}^{T_l-1} {\alpha^{l''}}_t(i) * a_{ij} * b_j(O_{t+1}) * {\beta^{l''}}_{t+1}(j))}{\sum_{l=1}^{K}\frac{1}{\prod_{k=1}^{T_l} {c^l}_k}(\sum_{t=1}^{T_l-1}\sum_{j=1}^{N} {\alpha^{l''}}_t(i) * a_{ij} * b_j({O^l}_{t+1}) * {\beta^{l''}}_{t+1}(j))}$$

$$\bar{\pi}_i = \frac{\sum_{l=1}^{K} \frac{1}{\prod_{k=1}^{T_l} c^l{}_k} \sum_{j=1}^{N} \alpha^{l''}{}_1(i) * a_{ij} * b_j(O_1) * \beta^{l''}{}_2(j)}{\sum_{l=1}^{K} \frac{1}{\prod_{k=1}^{T_l} c^l{}_k} \sum_{j=1}^{N} \alpha^{l''}{}_T(j)}$$

Nótese que el término $\frac{1}{\prod_{k=1}^{T_l} c^l{}_k}$ se introduce para eliminar el residuo de utilizar las variables *forward* y *backward* escaladas en el cálculo. Además entiéndase que hemos usado $\sum_{j=1}^{N} \alpha^{l''}{}_t(i) * a_{ij} * b_j(O_{t+1}) * \beta^{l''}{}_{t+1}(j)$ en lugar de $\alpha^{l''}{}_t(i) * \beta^{l''}{}_t(i)$ por razones de escalamiento ya que

$$\alpha^{l''}{}_t(i) * \beta^{l''}{}_t(i) = \prod_{k=1}^{t} c^l{}_k * \alpha^l{}_t(i) * \prod_{k=t}^{T} c^l{}_k * \beta^l{}_t(i)$$

$$= \prod_{k=1}^{T} c^l{}_k * c^l{}_t * \alpha^l{}_t(i) * \beta^l{}_t(i)$$

Haciendo imposible que el término $\frac{1}{\prod_{k=1}^{T} c^l{}_k}$ pueda eliminar los residuos del escalamiento. Mientras que $\sum_{j=1}^{N} \alpha^{l''}{}_t(i) * a_{ij} * b_j(O_{t+1}) * \beta^{l''}{}_{t+1}(j)$

$$= \sum_{j=1}^{N} \prod_{k=1}^{t} c^l{}_k * \alpha^l{}_t(i) * a_{ij} * b_j(O_{t+1}) * \prod_{k=t}^{T} c^l{}_k * \beta^l{}_{t+1}(j)$$

$$= \prod_{k=1}^{T} c^l{}_k * \alpha^l{}_t(i) * a_{ij} * b_j(O_{t+1}) * \beta^l{}_{t+1}(j)$$

Cumpliéndose el objetivo de eliminar los efectos secundarios del escalamiento y obteniendo los valores reales del proceso de reestimación. También es importante recalcar que las expresiones sustituidas son equivalentes ya que $\alpha_t(i) * \beta_t(i)$ denota la probabilidad de que el modelo genere la secuencia de observaciones cumpliendo que el estado i ocurra en el momento t. Mientras que $\sum_{j=1}^{N} \alpha_t(i) * a_{ij} * b_j(O_{t+1}) * \beta_{t+1}(j)$ es exactamente lo mismo lo que ahora hemos hecho explícito el paso del estado i a todas las posibles opciones en el momento t. Lo cual quedaba implícito en la primera expresión. Sin embargo, esta pequeña diferencia impide la duplicación del factor de escala del momento t.

Por último las funciones de reestimación para los parámetros de las mezclas serían de la siguiente forma

$$\bar{c}_{im} = \frac{\sum_{l=1}^{K} \sum_{t=1}^{T_l} \gamma^{l''}{}_t(i,m)}{\sum_{l=1}^{K} \sum_{t=1}^{T_l} \sum_{m=1}^{M} \gamma^{l''}{}_t(i,m)}$$

$$\bar{\mu}_{im} = \frac{\sum_{l=1}^{K} \sum_{t=1}^{T_l} \gamma^{l''}{}_t(i,m) * O^l{}_t}{\sum_{l=1}^{K} \sum_{t=1}^{T_l} \gamma^{l''}{}_t(i,m)}$$

$$\bar{U}_{im} = \frac{\sum_{l=1}^{K} \sum_{t=1}^{T_l} \gamma^{l''}{}_t(i,m) * (O^l{}_t - \mu_{im})^T (O^l{}_t - \mu_{im})}{\sum_{l=1}^{K} \sum_{t=1}^{T_l} \gamma^{l''}{}_t(i,m)}$$

La ausencia del factor $\frac{1}{\prod_{k=1}^{T} c^l{}_k}$ usado en las funciones de reestimación para las probabilidades de transición y de inicio se debe a que desde la variable $\gamma^{l''}{}_t(i,m)$ los residuos del escalamiento son eliminados como vimos antes en la sección de escalamiento. O sea los el escalamiento no afecta el resultado de los $\gamma^l{}_t(i,m)$ para cada una de las secuencias de entrenamiento. De esta forma las funciones de reestimación pueden incluir escalamiento e información proveniente de múltiples observaciones para entrenar los parámetros del modelo.

Capítulo 3: Implementación

Código

El proyecto se encuentra dividido en dos grupos de módulos: *logic* y *ui*. El primero se encarga del tratamiento y procesamiento de los datos, mientras que el segundo maneja la creación e interacción con la interfaz gráfica.

Los módulos que componen a *logic* son: *hmm.py*, *mixtures.py, utils.py*, *mfcc.py* y *sigproc.py*. Por su parte *ui* posee a: *main_window.py*, *dialogs.py* y *binding.py*. Veamos cada uno de ellos con más detalle.

hmm.py

En este módulo se encuentran las implementaciones más importantes del proyecto. Veamos la clase *HMM*:

```
class HMM:
def __init__(self, nstates, obsdim, nmixtcomp, smallervar=None):
    """Crea un objeto que representa un Modelo Oculto de Markov (HMM)

    :param nstates: Número de estados para el HMM.:param obsdim: Número de
      dimensiones de los vectores de observación.
    :param nmixtcomp: Si el parámetro es un int es usado para denotar el
      número de componentes para cada estado. De no ser un int,
      entonces debe ser una lista de números que contenga el número de
      componentes por cada estado.
    :param smallervar: Valor opcional que determina la varianza mínima
      permitida para las matrices de covarianza. En el caso de None es
      calculado determinando la raíz n-ésima de obsdim. Se recomienda mantener
      con su valor por defecto, si no se entiende correctamente su labor.
    :return Devuelve un HMM con las características especificadas.
    self.nstates = nstates
    self.obsdim = obsdim
    self.nmixtcomp = nmixtcomp

    if smallervar is None:
        self.smallervar = nroot(under_threshold, obsdim)
    else:
        self.smallervar = smallervar

    # Nótese que estos HMM son del tipo donde los procesos solo puede ir
    # hacia adelante (en este caso solo un paso) o hacia ellos mismos,
    # además de que inician en un estado predefinido, eliminando la
    # necesidad de una probabilidad inicial.
    self.transition_matrix = np.zeros((self.nstates, self.nstates))

    if type(nmixtcomp) is int:
        self.states = np.array([mx.Mixture(nmixtcomp, obsdim) for
                                i in range(nstates)])
    elif type(nmixtcomp) is list and len(nmixtcomp) ==
              nstates:
        self.states = np.array([mx.Mixture(nmixtcomp[i], obsdim)
                                for i in range(nstates)])
    else:
        raise ValueError("El parámetro 'nmixtcomp' debe ser un 'int'
                         o una tupla de longitud igual al parámetro
                         'nstates'.")
```

```
def forward_algorithm(self, training_seq):
    """Computa las variables forward para la secuencia dada.

    :param training_seq: Secuencia de observaciones de la cual se extraen los
      valores de las variables forward.
    :return: Una matriz de variables forward escaladas y una lista que
      contienen los escaladores.
    """
    length_seq = len(training_seq)
    scale_factors = [0.] * length_seq
    scale_factors[0] = self.states[0].get_density(training_seq[0])

    forward_vars = np.ones((self.nstates, length_seq)) * under_threshold
    forward_vars[0, 0] = self.states[0].get_density(training_seq[0])
                               / scale_factors[0]

    for curr_time in range(1, len(training_seq)):
        for curr_state in range(self.nstates):
            # Calcula el alfa(curr_state, curr_time). Esto es la
            # variable forward en el tiempo 'curr_time' para el tiempo
            # 'curr_state'.
            forward_vars[curr_state, curr_time] = sum(
                [forward_vars[i, curr_time - 1] *
                self.transition_matrix[i, curr_state] for i in
                range(self.nstates)]) *
                self.states[curr_state].get_density(training_seq[curr_time
                ])

            forward_vars[curr_state, curr_time] =
                _fix_threshold(forward_vars[curr_state, curr_time],
                under_thd=under_threshold, upper_thd=upper_threshold)

        # Salvando el factor de escalamiento 'curr_time' para luego ser
        # usado en el cómputo de las variables backwards.
        scale_factors[curr_time] = sum(forward_vars[:, curr_time])
        # Aplicando el factor de escalamiento.
        forward_vars[:, curr_time] /= scale_factors[curr_time]

    return forward_vars, scale_factors

def backward_algorithm(self, training_seq, scale_factors):
    """Computa las variables backward para la secuencia dada.

    :param training_seq: Secuencia de observaciones de la cual se extraen
      los valores de las variables forward.
    :param scale_factors: Secuencia de escaladores a emplear en el cómputo de
      las variables backward.
    :return: Una matriz de variables backward escaladas.
    """

    length_seq = len(training_seq)

    backward_vars = np.zeros((self.nstates, length_seq))
    backward_vars[:, -1] = 1. / scale_factors[-1]

    for curr_time in range(length_seq - 2, -1, -1):
        for curr_state in range(self.nstates):
            # Calcula beta(curr_state, curr_time). Esto es la variable
            # backward en tiempo 'curr_time' para el estado 'curr_state')
            backward_vars[curr_state, curr_time] =
                sum([self.transition_matrix[curr_state, i] *
                self.states[i].get_density(training_seq[curr_time + 1]) *
                backward_vars[i, curr_time + 1] for i in
                range(self.nstates)])
```

```
            backward_vars[curr_state, curr_time] =
                _fix_threshold(backward_vars[curr_state, curr_time],
                under_thd=under_threshold, upper_thd=upper_threshold)

        backward_vars[:, curr_time] /= scale_factors[curr_time]
        backward_vars[:, curr_time][backward_vars[:, curr_time] ==
        float('inf')] = upper_threshold

    return backward_vars

def viterbi_algorithm(self, sequence, path=False):
    """Realiza el algoritmo viterbi sobre la secuencia dada.

    :param sequence: Secuencia de observaciones sobre la que realizar el
      cómputo.
    :param path: Valor opcional que determina si computar el camino del
      algoritmo viterbi o no.
    :return: Retorna la probabilidad de emisión de la mejor secuencia de
      estados para el modelo. Si path es True, entonces también devuelve una
      lista con los estados del camino a seguir para esta.
    """
    length_seq = len(sequence)

    lasts_vit_vars = [float('-inf')] * self.nstates
    lasts_vit_vars[0] = log10(self.states[0].get_density(sequence[0]))

    if compute_path:
        # Contiene en la celda (i, j) el estado que precede al estado i
        # en el camino óptimo de longitud j que terminan en i.
        optimal_vars = np.zeros((self.nstates, len(sequence)),
                                        dtype=int)

    for index_obs in range(1, length_seq):

        new_vit_vars = [0.] * self.nstates
        for i in range(self.nstates):
            # Almacena todas las posibles transiciones a 'curr state'
            # desde los anteriores para luego elegir el mejor.
            transitions = [lasts_vit_vars[j] +
                self.log_transition_matrix[j][i] for j in
                range(self.nstates)]

            new_vit_vars[i] = max(transitions) +
                log10(self.states[i].get_density(sequence[index_obs]))

            if compute_path:
               optimal_vars[i, index_obs] =
                        transitions.index(max(transitions))

        lasts_vit_vars = new_vit_vars

    # Obteniendo el camino óptimo
    if compute_path:
        optimal_path = np.zeros(length_seq)
        optimal_path[-1] = lasts_vit_vars.index(max(lasts_vit_vars))

        for i in range(length_seq - 1, 0, -1):
            optimal_path[i - 1] = optimal_vars[optimal_path[i], i]

        return max(lasts_vit_vars), optimal_path
    else:
        return max(lasts_vit_vars)
```

```
def estimate_parameters(self, seg_obs, ntrainingseqs=1, oldcentroids=False):
    """Estima los parámetros del modelo a partir de una secuencia de
    observaciones segmentada.

    Usando el método k-means este procedimiento obtiene “nmixtcomp” subgrupo
    de cada grupo de datos en “seg_obs”. Luego de cada grupo extrae
    información con la cual inicializa al modelo. Los coeficientes de las
    mezclas son obtenidos a través de la importancia de cada grupo relativa al
    resto. Las varianzas son extraídas de las varianzas de cada grupo. Las
    medias son obtenidas directamente desde el k-means. Las transiciones son
    también estimadas de manera similar a los coeficientes.

    :param seg_obs: Lista que contiene las observaciones que pertenecen a
      cada estado.
    :param ntrainingseqs: Valor opcional que especifica el número de
      secuencias de observaciones suministradas durante el entrenamiento. El
      valor por defecto es 1.
    :param oldcentroids: Valor opcional que determina si nuevas medias deben
      ser computadas por el k-means o se deben emplear las del modelo en
      cuestión. Su valor por defecto es calcula nuevas medias, así que de no
      entender el funcionamiento de este parámetro mantenga el valor por
      defecto.
    :return: No retorna nada
    """
    for i in range(self.nstates):
        if len(seg_obs[i]) == 0:
            # Algunas veces el clustering de los estados deja alguno de
            # ellos vacío, por lo que sería imposible estimar algo para
            # dicho estado. Así que simplemente se inicializan sus
            # parámetros de manera que sea tan improbable de suceder
            # como sea posible. Nótese que esto es diferente a las
            # componentes vacías, las cuales pueden ser simplemente
            # eliminadas.
            self.states[i].return_minimum_density = True

            self.transition_matrix[i, i] = under_threshold
            if i < self.nstates - 1:
                self.transition_matrix[i, i + 1] = 1.

            continue

        # Salvando la desviación estándar de cada característica para
        # recuperar más tarde sus verdaderas medias, luego de aplicar
        # whitenning.
        standard_deviations =
              [msr.standard_deviation(seg_obs[i][:, j]) for j in
               range(self.obsdim)]

        # A veces el procedimiento kmeans2 lanza una excepción
        # relacionda con la imposibilidad de realizar una descomposición
        # de Cholesky sobre los datos.Esto se debe a una cantidad de
        # observaciones para el estado estado más pequeña que el número
        # de componentes, por lo que usamos un método de inicialización
        # del kmeans diferente.
        try:
            if not oldcentroids:
                # Llevando las características de las observaciones a
                # varianza unitaria. Lo cual mejora el resultado del
                # kmeans.
                means, labels = vq.kmeans2(vq.whiten(seg_obs[i]),
                      self.states[i].ncomponents, minit='random')
            else:
                means, labels = vq.kmeans2(vq.whiten(seg_obs[i]),
                      np.array([[float(feature) /
```

```
                standard_deviations[index] for index, feature in
                enumerate(oldcentroids[i][j][0])] for j in
                range(self.states[i].ncomponents)]),
                minit='matrix')
except np.linalg.linalg.LinAlgError:
    # Si hubo un problema, entonces utilizo centroides
    # inicialies iguales a valores en los datos.
    means, labels = vq.kmeans2(vq.whiten(seg_obs[i]),
                self.states[i].ncomponents, minit='points')
# Arreglando las medias (revirtiendo los efectos del whitenning)
means[:] *= standard_deviations

# Remover los centroides cuyos clústeres no tengan elementos.
means, labels = delete_clusters_less_than(seg_obs[i], means,
                                          labels)

# Actualizar el número de componentes, pues alguna pudo ser
# eliminada en el paso anterior.
self.states[i].ncomponents = len(means)

self.states[i].components = []
self.states[i].coefficients = []

for component in range(self.states[i].ncomponents):
    # Cuento cuantas observaciones pertenecen al cluster 'i' y
    # establezco su coeficiente como el radio entre este número
    # y el número de observaciones.
    self.states[i].coefficients.append((labels ==
                component).sum() / float(len(labels)))

    # Obtengo las varianzas asociadas con cada dimendion de los
    # vectores asociados con cada componente. Con esto obtengo
    # una matriz diagonal con las varianzas pertenecientes a un
    # clúster. Nótese como no se aceptan valores de varianza
    # menor o igual a cero (lo cual puede pasar si el número de
    # observaciones en un clúster es de solo uno y por tanto
    # esta corresponde con el centroide). La razón de esto se
    # debe a que más adelante se necesitará obtener la inversa
    # de esta matriz y su determinante no puede ser cero.
    if np.count_nonzero(labels == component) > 1:
        cov_matrixes =
                np.diag([nd.variance(seg_obs[i][labels ==
                component][:, j]) for j in range(self.obsdim)])
    else:
        cov_matrixes = np.diag([one_member_cluster_variance] *
                self.obsdim)

    # Creo las 'ncomponents' de la mezcla. Observese que usamos
    # distribuciones normales multivariadas para las
    # componentes, aunque podemos usar cualquier función con
    # dendidad elípticamente simétrica.
    self.states[i].components.append(
                mx.MultiNormalDist(means[component],
                                   cov_matrixes))

# Cuenta cuantas observaciones pertenecen a un estado dado y
# estima el radio de saltos hacia él mismo, siendo solo una
# transición para saltar al próximo estado. Nótese que el último
# estado permanece en sí mismo.
if i != self.nstates - 1:
    transitions_in_i = len(seg_obs[i])
    transitions_out_i = ntrainingseqs
    total_transitions = float(transitions_in_i +
                              transitions_out_i)
```

```
                self.transition_matrix[i, i] = transitions_in_i /
                        total_transitions
                self.transition_matrix[i, i + 1] = transitions_out_i /
                        total_transitions
            else:
                self.transition_matrix[i, i] = 1

            self.states[i].return_minimum_density = False

        # Computa y almacena ciertos valores usados con frecuencia en el
        # modelo como: la matriz de transición en escala logarítimica, la
        # inversa de matriz de transición para cada una de sus componentes,
        # así como la primera parte de las fórmulas de densidad en ellas,
        # las cuales no dependen del evento, y por tanto, son siempre la
        # misma si la media y la matriz de covarianza no cambian.
        self.end_building()
```

Además de la clase *HMM*, este módulo también posee algunas funciones de vital importancia.

```
def state_segmentation(hmm, training_seqs, method='viterbi'):
    """Segmenta múltiples (o solo una) secuencia(s) de observaciones en
        tantos grupos como estados tenga el modelo especificado.

    :param hmm: Modelo del cual se extrae la información requerida durante la
        segmentación.
    :param training_seqs: Lista de secuencia(s) de observaciones.
    :param method: Valor opcional que determina el método empleado para
        segmentar. Los posibles valores son "viterbi" o "uniforme". El
        primero calcula el mejor camino para cada secuencia y clasifica
        sus observaciones acorde al estado subyacente en la cadena,  mientras
        que "uniforme" asigna a cada estado la misma cantidad        de
        observaciones sin realizar ningún procesamiento extra.
    :return: Una lista con las secuencias de observaciones que pertenecen a
        cualquier estado dado.
    """
    if multiple:
        result = None
        for sequence in training_seqs:
            segm_states = single_state_segmentation(hmm, sequence, method)
            if result is None:
                result = segm_states
            else:
                for i in range(hmm.nstates):
                    result[i] = np.concatenate((result[i], segm_states[i]))

        return result

    else:
        return single_state_segmentation(hmm, training_seqs, method)
```

La anterior función se apoya en la siguiente:

```
def single_state_segmentation(hmm, training_seq, method='viterbi'):
    """Segmenta una única secuencia de observaciones en grupos de estados.

    :param hmm: Modelo a partir del cual se extraerá la información requerida
        para efectuar la segmentación.
    :param training_seq: Secuencia de observaciones a segmentar.
    :param method: Determina el método a emplear para la segmentación.
    :return: Una lista de secuencia de observaciones donde cada lista
        superior corresponde a las observaciones del estado
        correspondiente a su índice.
    """
```

```
    sequence = np.array(training_seq)
    length_seq = len(sequence)

    # Si solo existe un estado, no hay necesidad de ningún cálculo extra.
    if hmm.nstates == 1:
        segm_states = [sequence]

    elif str.lower(method) == 'uniform':
        sizes = np.array([length_seq // hmm.nstates] * hmm.nstates)
        sizes[r.sample(range(hmm.nstates), length_seq % hmm.nstates)] += 1

        segm_states = [sequence[sum(sizes[: i]):sum(sizes[: i + 1])] for i
                       in range(hmm.nstates)]

    elif str.lower(method) == 'viterbi':
        prob, opt = hmm.viterbi_algorithm(sequence, compute_path=True)
        segm_states = [sequence[opt == i] for i in range(hmm.nstates)]

    else:
        raise ValueError("El parámetro 'method' debe contener solo estas
                         opciones: 'viterbi' or 'uniform'.")

    return segm_states
```

Por último veáse la implementación del método de reestimación de Baum-Welch:

```
def baum_welch_reestimation (curr_model, training_seqs):
    """Aplica las funciones de reestimación del algoritmo Baum-Welch para el
    modelo dado, empleando las secuencias de observaciones dadas.

    :param curr_model: Modelo al cual reestimar sus parámetros
    :param training_seqs: Las secuencia(s) de observaciones empleadas en la
        reestimación.
    :return: Un nuevo modelo con los parámetros actualizados mediante el
        procedimiento.
    """
    new_model = HMM(curr_model.nstates, curr_model.obsdim,
                    nmixtcomp=[curr_model.states[i].ncomponents for i
                               in range(curr_model.nstates)],
                    smaller_var=curr_model.smaller_var)

    decimal_transition_matrix = np.zeros((curr_model.nstates,
    curr_model.nstates), dtype=d)

    for sequence in training_seqs:
        length_seq = len(sequence)

        forward_vars, scale_factors =
                        curr_model.forward_algorithm(sequence)
        backward_vars = curr_model.backward_algorithm(sequence,
                                                      scale_factors)

        # Used to cancel the effect of the scaling procedure
        scale_factors_prod = prod(scale_factors, dtype=d)

        for i in range(curr_model.nstates):
            # Calculo el numerado de la fórmula de reestimación de la
            # probabilidad de transición.
            decimal_transition_matrix[i, i] += scale_factors_prod *
                sum([d(forward_vars[i, time]) *
                        d(curr_model.transition_matrix[i, i]) *
                        d(curr_model.states[i].get_density(sequence[time +
                        1])) * d(backward_vars[i, time + 1]) for time in
                        range(length_seq - 1)])
```

```
            if i < curr_model.nstates - 1:
                decimal_transition_matrix[i, i + 1] += scale_factors_prod
                        * sum([d(forward_vars[i, time]) *
                        d(curr_model.transition_matrix[i, i + 1]) *
                        d(curr_model.states[i +
                        1].get_density(sequence[time + 1])) *
                        d(backward_vars[i + 1, time + 1]) for time in
                        range(length_seq - 1)])

            for component in range(curr_model.states[i].ncomponents):
                # Esta lista termina conteniendo las probabilidades de
                # estar en el estado 'i' acorde a la componente 'm' de su
                # mezcla en un momento dado 't'.
                gamma_i_comp = []
                for time in range(length_seq):
                    gamma_i_comp.append(forward_vars[i, time] *
                        backward_vars[i, time] / sum([forward_vars[state,
                        time] * backward_vars[state, time] for state in
                        range(curr_model.nstates)]) *
                        (curr_model.states[i].get_density(sequence[time],
                        component=component) /
                        curr_model.states[i].get_density(sequence[time])))

                    gamma_i_comp[-1] = _fix_threshold(gamma_i_comp[-1],
                        under_thd=under_threshold,
                        upper_thd=upper_threshold, nan=under_threshold)

                    # Computo los numeradores de las fórmulas de
                    # reestimacion de los parámetros de las componentes de
                    # las mezclas.
                    new_model.states[i].coefficients[component] +=
                        gamma_i_comp[time]
                    new_model.states[i].components[component].mean +=
                        gamma_i_comp[time] * sequence[time]

new_model.states[i].components[component].covariance_matrix +=
    gamma_i_comp[time] * np.dot((sequence[time] -
    curr_model.states[i].components[component].mean).T,
    sequence[time] - curr_model.states[i].components[component].mean)

# Añado los denominadores a las fórmulas de reestimación.
for state in range(new_model.nstates):
    prob_states_sum = sum(decimal_transition_matrix[state])
    # A veces cuando se posee poca información y muchos estados la
    # fórmula de reestimación da cero a todos los valores en un mima
    # fila. En tales casos asumimos que el estado es solo transitorio
    # y debemos salir de él tan pronto como sea posible.
    if prob_states_sum == 0:
        if state != new_model.nstates - 1:
            decimal_transition_matrix[state, state + 1] = d(1)
        else:
            decimal_transition_matrix[state, state] = d(1)
    else:
        # Computo y añado el denominador a la fórmula de reestimación
        # para la transición y así obtengo los valores reestimados.
        decimal_transition_matrix[state] /= prob_states_sum

    for component in range(new_model.states[state].ncomponents):
        # Hago lo mismo para las fórmulas de los parámetros de las
        # mezclas. Nótese como los denominadores para la media y la
        # matriz de covarianza es el numerador de la fórmula de
        # reestimación de los coeficientes.
        new_model.states[state].components[component].mean /=
            new_model.states[state].coefficients[component]
```

```
                new_model.states[state]
                    .components[component].covariance_matrix /=
                    new_model.states[state].coefficients[component]

            new_model.states[state].coefficients /=
                    sum(new_model.states[state].coefficients)

        # Salvo los valores computados para la matriz de transición.
        new_model.transition_matrix = decimal_transition_matrix.astype(float)
        # Arreglo las probabilidades que son muy pequeñas o cero, dado que
        # esto puede prevenir el paso a través de los estados.
        for i in range(new_model.nstates):
            new_model.transition_matrix[i, i] =
                    _fix_threshold(new_model.transition_matrix[i, i],
                    under_thd=under_threshold, nan=under_threshold)

            if i < new_model.nstates - 1:
                new_model.transition_matrix[i, i + 1] =
                    _fix_threshold(new_model.transition_matrix[i, i + 1],
                    under_thd=under_threshold, nan=under_threshold)

        new_model.end_building()
        _fix_singular_matrixes(new_model, curr_model)

        return new_model
```

Por último se encuentran las clases: *TrainModel* y *RecognizeModel* que permiten realizar las operaciones de entrenamiento y reconocimiento acoplando todos los procedimientos antes mencionados de la forma correcta.

mixtures.py

Aquí se implementan las clases: *Mixture* y *MultiNormalDist* que se encargan de representar las mezclas gaussianas y sus componentes. Sus códigos se encuentran a continuación.

```
class Mixture:
    def __init__(self, ncomponents, ndimensions):
        """Crea un objeto que representa un combinación lineal de distribuciones
        de densidad.

        :param ncomponents: Número de componentes en la mezcla.
        :param ndimensions: Número de dimensiones para las componentes de la
                mezcla.
        :return: Un objeto que representa la mezcla especificada.
        """
        # Este atributo es usado en caso que queramos que la mezcla retorne
        # un valor pequeño prefijado, lo cual es útil durante la estimación
        # cuando ocurran estados vacíos.
        self.return_minimum_density = False

        self.ncomponents = ncomponents
        self.ndimensions = ndimensions

        self.coefficients = [0.] * ncomponents

        # Crea las 'ncomponents' de la mezcla. Esto podría ser cualquier
        función con densidad elpticamenten simétrica, pero en este caso
        utilizamos la distribución normal multivariada.
        self.components = [MultiNormalDist([0.] * ndimensions,
                        np.zeros((ndimensions, ndimensions))) for i in
                        range(ncomponents)]
```

```
    def get_density(self, event, component=None):
        """Computa la densidad del evento pasado acorde a la mezcla.

        Suma las densidades de cada una de las componentes de la mezcla
        multiplicadas por su respectivo coeficiente.

        :param event: Vector cuya densidad se desea computar.
        :param component: Especifica la componente de la mezcla en cuya
                densidad estamos interesados. En caso de 'None' (valor por
                defecto) se suman todas.
        :return: La densidad del vector 'event'.
        """
        # A veces aun cuando no hay observaciones para un estado, y por
        # tanto sus parámetros no pueden ser estimadis, este todavía es
        # consultado sobre diversas densidades. Es por eso que
        # establecimos un valor extremadamente bajo para cualquier
        # observación si el estado se encuentra en esta situación.
        if self.return_minimum_density:
            return under_threshold

        if component is None:
            res = np.sum([self.coefficients[i] *
                self.components[i].get_density(event) for i in
                range(self.ncomponents)])
        elif component >= self.ncomponents or component < 0:
            raise ValueError('Invalid component value.')
        else:
            res = self.coefficients[component] *
                        self.components[component].get_density(event)

        # Asegurándome de que la densidad se encuentre dentro de los
        # umbrales establecidos. O sea que no exista ningún underflow u
        # overflow.
        return hmm._fix_threshold(res, under_thd=under_threshold,
                                  upper_thd=upper_threshold)


class MultiNormalDist:
    def __init__(self, mean, covariance_matrix):
        """Crea un objeto que representa una distribución normal multivariada.

        :param mean: Un array que representa el vector de la media en la
                distribución.
        :param covariance_matrix: Un array que representa la matriz de covarianza
                de la distribución. Esta debe cumplir que su determinante sea
                mayor estricto que cero.
        :return: Un objeto que representa la distribución normal multivariada
                especificada.
        """
        self.ndimensions = len(mean)
        # Convierte los valores de entrada a matrices para operaciones
        # posteriores.
        self.mean = np.array([mean])
        self.covariance_matrix = covariance_matrix
```

```
    def get_density(self, event):
        """Computa la densidad de 'event' acorde a la instancia de
            'MultiNormalDist'.

            :param event: El vector cuya densidad se desea calcular.
            :return: La densidad del vector 'event'.
            """
            if self.ndimensions != len(event):
                raise ValueError('Vector\'s dimension different from the
                        distribution\'s dimension')

        # Convierte los valores de entrada a matrices para operaciones
        # posteriores.
        m_event = np.array([event])
        return self.first_part_of_formulae * (math.e ** (-0.5 *
                np.dot(np.dot(m_event - self.mean,
                self.inv_covariance_matrix), (m_event -
                self.mean).T)[0,0]))
```

utils.py

Como su nombre lo indica, aquí se encuentran un grupo de funciones que sirven de apoyo al resto de los módulos en *logic*. Entre las más importantes encontramos:

```
def delete_clusters_less_than(data, means, labels, less_than=1):
    """Elimina todos los grupos cuyo tamaño sea menor estricto que un valor
    especificado.

    :param data: Secuencia original sobre la que se calcularon los grupos.
    :param means: Medias de los distintos grupos encontrados.
    :param labels: Lista que especifica mediante un entero el grupo al cual
        pertenece el valor en 'data' que se encuentre en la misma posición.
    :param less_than: Valor opcional que determina el tamaño de los grupos a
        eliminar.
    :return: Un par 'means' y 'labels', actualizados luego de retirar los grupos
        requeridos.
    """


def nroot(x, n):
    """Calcula la n-ésima raíz de un número positivo.

    :param x: Número a encontrar su raíz.
    :param n: Profundidad de la raíz.
    :return: La n-ésima raíz de 'x'.
    """
    return m.exp(m.log(x) / float(n))
```

Además encontramos otras funciones como *log10*, que aplica la operación logarítmica típica, pero con la salvedad de que al evaluar cero retorna *–inf* o también *prod* que retorna el producto de una secuencia, pero convirtiendo cada elemento a un tipo previamente especificado.

mfcc.py

En este módulo encontramos las funciones relacionadas con la obtención de los Coeficientes Cepstrales de Mel.

```
def mfcc(signal, samplerate=16000, winlen=0.025, winstep=0.01, numcep=13,
         nfilt=26, nfft=512, lowfreq=0, highfreq=None, appendEnergy=True):
    """Computa los MFCC a partir de una señal de audio.

    :param signal: La señal de audio a partir de la cual extraer las
         características.
    :param samplerate: La frecuencia de muestreo que estemos usando.
    :param winlen: La longitud de la ventana de análisis en segundos.
    :param winstep: El tamaño del paso, en segundos, entre ventanas sucesivas
    :param numcep: El número de valores cepstrales a devolver.
    :param nfilt: El número de filtros en el banco de filtros de Mel.
    :param nfft: El tamaño de la FFT.
    :param lowfreq: Banda mínima de frecuencias para los filtros de Mel en Hz.
    :param highfreq: Banda máxima de frecuencias para los filtros de Mel en Hz.
         De ser omitido es 'samplerate / 2'.
    :param appendEnergy: Si esto es True el primer coeficiente cepstral es
         reemplazado con el logaritmo de la energía total de la ventana.
    :return: Un array que contiene las características.
    """
    feat, energy = fbank(signal, samplerate, winlen, winstep, nfilt, nfft,
                         lowfreq, highfreq, preemph)
    # Reemplazo todos los valores '-inf' en los coeficientes debido a los
    logaritmos.
    feat[feat == 0] = under_threshold
    feat = numpy.log(feat)
    feat = dct(feat, type=2, axis=1, norm='ortho')[:, :numcep]
    feat = lifter(feat, ceplifter)
    if appendEnergy:
        # Reemplaza el primer coeficiente cepstral con el logarithm of the
        # enegy.
        feat[:, 0] = numpy.log(energy)

    return feat
```

También existen otras funciones importantes como: *get_filterbanks* que computa los filtros de Mel o *hz2mel* y *mel2hz* que permiten convertir valores de Hz a Mel y viceversa.

sigproc.py

En este módulo se encuentra *framesig* que permite la fragmentación de la señal en ventanas:

```
def framesig(sig, frame_len, frame_step, winfunc=lambda x:numpy.ones((1,x))):
    """Segmenta una señal en fragmentos superpuestos.

    :param sig: La señal de audio a segmentar.
    :param frame_len: Longitud de cada fragmento medido en muestras.
    :param frame_step: Número de muestras, luego del inicio del fragmento
         anterior en que debe empezar el próximo.
    :param winfunc: La función de alisamiento aplicada a cada fragmento.
    :returns: Un array de fragmentos.
    """
```

Además, también se encuentran las funciones *magspec* y *powspec* que realizan, respectivamente, el cómputo de la TDF y el cálculo del periodograma para los fragmentos en que se dividió la señal.

Entrenamiento de un modelo

Para entender cómo se realiza el proceso de entrenamiento de un modelo, veamos la forma en que se utilizan cada uno de los recursos anteriores si asumimos que se posee un conjunto de N archivos de audios pertenecientes a un mismo animal y se le ordena al sistema que construya un reconocedor.

Inicialmente la aplicación extraerá todos los fragmentos posibles para cada uno de los audios y por cada uno de ellos computará el MFCC correspondiente. Esto se traduce en una llamada a *mfcc* por cada uno de los archivos. Al terminar se tendrán N secuencias de MFCCs donde cada una corresponde a un archivo determinado.

A continuación se dividen uniformemente cada una de las secuencias en el número de estados. Esto quiere decir que si se ha especificado construir un reconocedor con S estados, entonces cada secuencia de MFCCs se dividirá en S grupos uniformes. La razón detrás de esto es que se ha asumido, a falta de más datos, que cada estado posee la misma importancia y por tanto, la única forma en que el reconocedor haya generado dicha señal es si cada uno de sus estados emitiese L_i/S de ella, donde L_i es la i-esima secuencia con $1 \leq i \leq N$.

Una vez fragmentada cada secuencia se aúnan los vectores que pertenecen a estados iguales sin importar la secuencia. Esto se realiza porque como cada secuencia fue emitida por el mismo reconocedor, entonces los grupos de vectores en que fueron separadas no son más que una ocurrencia del estado correspondiente, razón por la cual se pueden utilizar cada uno de estos vectores para inferir la distribución del estado.

Recuérdese que para modelar la distribución de probabilidad de los estados se decidió emplear las mezclas gaussianas. Estas se encuentran formadas por varias componentes y cada una corresponde a un grupo distinto de vectores. Nótese que los grupos a los que nos referimos son a los grupos en los que se dividen los vectores que pertenecen a un mismo estado. Por lo que si el usuario ha especificado M componentes por mezcla, entonces se intentará agrupar los vectores en M grupos.

La función *estimate_parameters* utiliza estos grupos y calcula sus medias y varianzas. Luego, al realizar esta operación para cada uno de los estados se obtienen parámetros iniciales para cada una de sus componentes. En cuanto a las transiciones, supongamos que se desea estimar el valor de la transición del estado S_1 al S_2, entonces la función halla la razón entre la cantidad de veces que dos vectores contiguos pertenecen a S_1 y a S_2 y la cantidad de vectores que pertenecen a S_1.

Una vez que se han estimado valores para el modelo se vuelve a segmentar, pero ahora se utiliza el algoritmo *viterbi* por cada secuencia de MFCCs. Esto nos permitirá determinar cuál es la secuencia de estados que más probablemente haya emitido dicha secuencia de observaciones (MFCCs) para el modelo actual. Entonces de la misma manera que antes, volvemos a unir todos los vectores pertenecientes a estados iguales y repetimos el proceso de estimación.

Como rasgo adicional nuestra aplicación realiza un proceso que denominamos *smoothing*, mediante el cual, cada cierto tiempo, después de una buena estimación y antes de volver a segmentar, se realizan pequeños cambios en los valores de las medias y varianzas del modelo. Experimentalmente hemos constatado que en numerosas ocasiones esto lleva a un pequeño incremento en la correctitud de los parámetros.

Una cuestión que no ha sido explicada es cómo comparar un modelo con respecto a otro. Para hacer esto se emplea el algoritmo *forward*, que devuelve $P(O|\lambda)$ donde O es la secuencia de observaciones y λ el modelo. También se puede utilizar el algoritmo *viterbi* para realizar esta comparación, lo que en esta ocasión en vez de comparar los $P(O|\lambda)$ de cada modelo se usaría la probabilidad de la mejor secuencia de estados que emita a O.

Después de repetido este proceso de segmentación y estimación un determinado número de veces y se cuente con un buen modelo para los audios, entonces se aplica el procedimiento de Baum-Welch (BW). Esto se debe a que el algoritmo es muy dependiente del modelo de entrada [23] no obstante, una vez aplicado, su $P(O|\lambda)$ será igual o superior a la del modelo obtenido mediante segmentación [16]. Sin embargo, se debe resistir la tentación de repetir esta operación y alimentar al BW con el resultado del anterior, pues provocaría sobreentrenamiento. Esto significa que el modelo se adaptaría tan bien al conjunto de entrenamiento que no arrojaría buenos resultados con otros archivos, aunque sean del mismo tipo.

Para evitar este problema en vez de ir directamente hacia el próximo BW, hemos aplicado un paso de estimación justo después del BW, donde segmentamos con el modelo generado por él y luego, usando el resultado de esta estimación, iniciamos la próxima iteración del BW.

Reconocimiento de un modelo

De manera similar al proceso de entrenamiento una vez se tiene el archivo a reconocer se extrae la secuencia de MFCCs correspondiente. Con esta se realiza el cálculo del algoritmo *forward* para cada modelo en comparación. Aquel que posea la $P(O|\lambda)$ más alta es el devuelto. Al igual que antes esta comparación también puede ser realizada usando el algoritmo *viterbi*, donde aquella secuencia con mayor probabilidad es la que determina el resultado.

main_window.py

En este módulo se implementa la interfaz gráfica de la aplicación como se muestra en la Fig. 14.

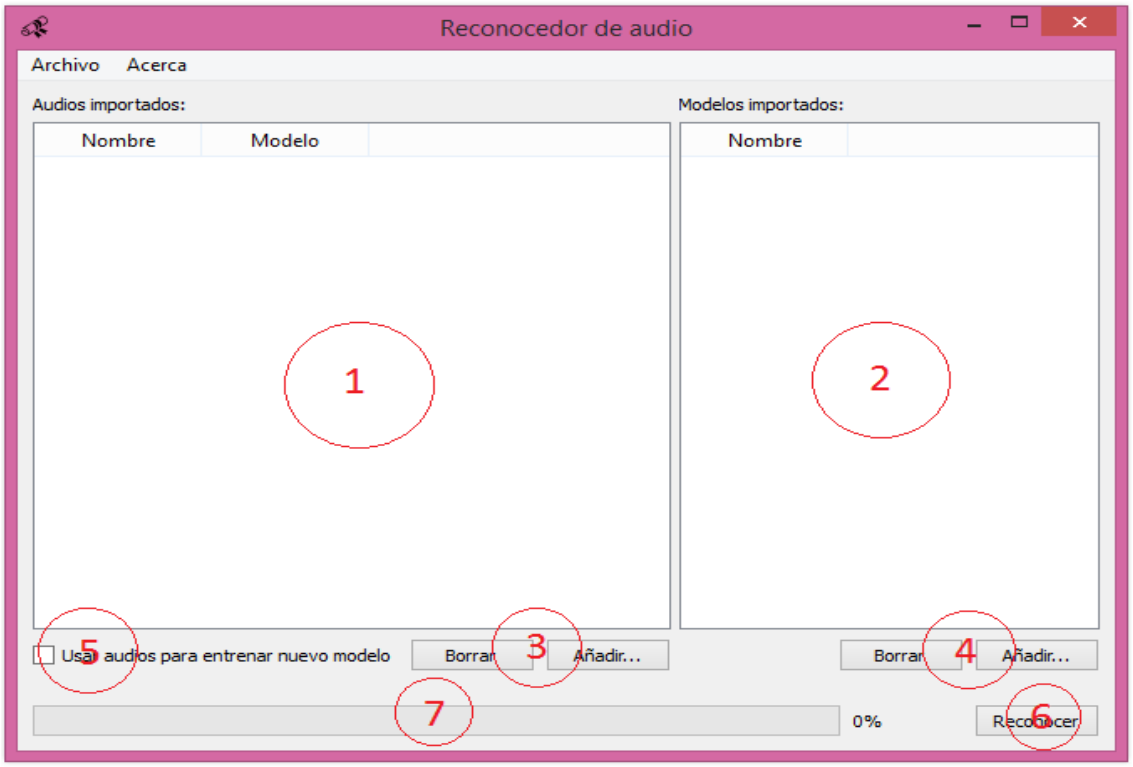

Fig. 14 Ilustración de la interfaz de la ventana principal.

En esta las zonas 1 y 2 corresponden, respectivamente, a los audios y modelos cargados actualmente. Mientras que 3 y 4 son las acciones de añadir y borrar archivos en estas. Además con 6 se da inicio al reconocimiento, aunque deben existir al menos un archivo y un modelo cargados para realizarlo.

La opción en 5 (vea la Fig. 15) permite usar los archivos en 1 para entrenar un nuevo modelo. Su elección provoca la desaparición de 2 y 4, mientras que 6 cambia su nombre a “Entrenar…”, representando la nueva acción que realiza.

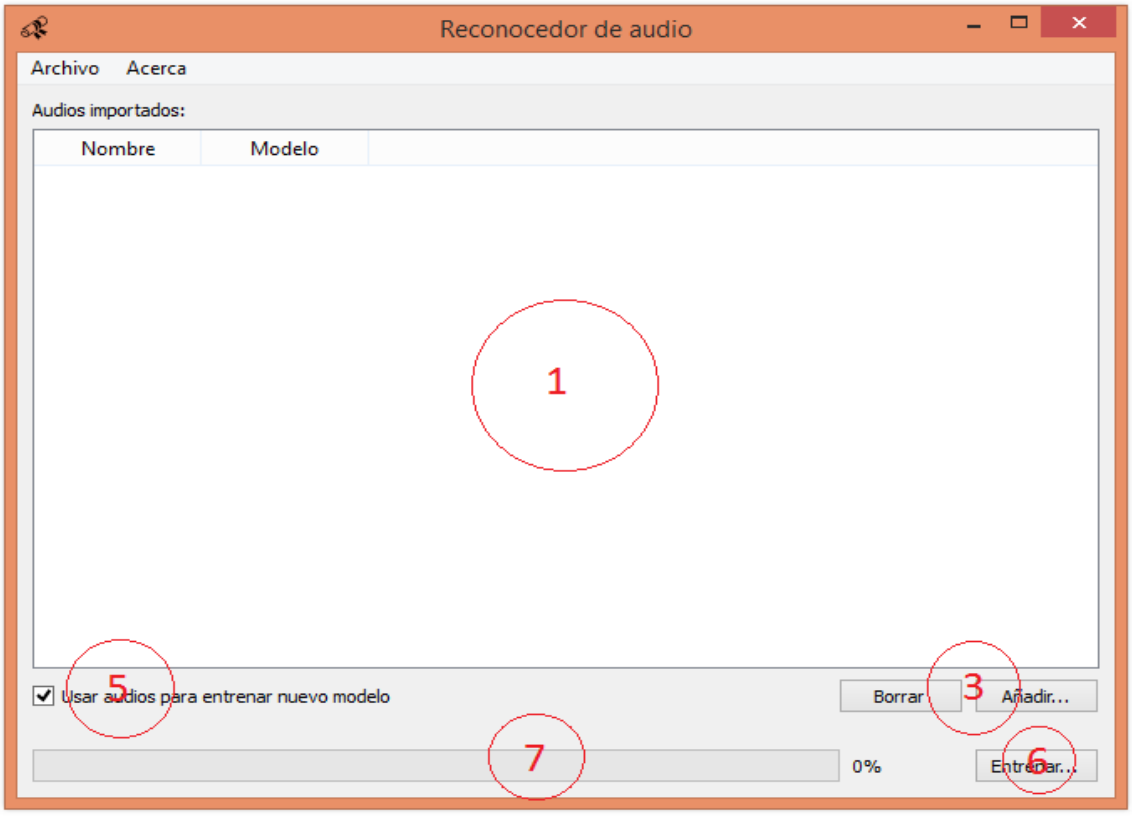

Fig. 15 Ilustración de la ventana principal en modo entrenamiento.

Si con 5 marcado se añade al menos un archivo y se presiona 6, entonces aparece un cuadro de diálogos (Fig. 17), que nos permite establecer las propiedades del modelo a entrenar (en el próximo epígrafe se verá esto con más detalle). En cualquiera de los casos , ya sea entrenamiento o reconocimiento, en 7 siempre se mostrará el avance de la operación elegida (Fig. 16) y al concluir esta se le notificará al usario mediante un cuadro de diálogo. Nótese además que 6 cambia a “Cancelar” permitiendo detener el proceso en cualquier momento, no obstante la aplicación le pedirá confirmación para realizar esta acción, en caso de que su elección no haya sido intencional.

Fig. 16 Ilustración de la ventana principal durante un procesamiento.

dialogs.py

En este módulo se especifican tres ventanas del tipo cuadro de diálogo:

- *WarningDlg*: Es la clase que permite crear los distintos mensajes de error al intentar operaciones inválidas o se complete alguna operación.
- *AboutDlg*: Ofrece información acerca de la aplicación, su propósito y creadores.
- *OptionDlg*: Ofrece un conjunto de características necesarias para la creación de nuestro modelo (Fig. 17). Estas son:
    - Dirección para almacenar el nuevo modelo creado.
    - Número de estados que deberá tener el modelo.
    - Número de componentes que deberá tener el modelo.
    - Número de iteraciones del proceso de segmentación-estimación.
    - Si aplicar o no el procedimiento Baum-Welch[16]
    - Número de iteraciones para el Baum-Welch si se decide aplicarlo.

[16] Podría desearse no aplicarlo para evitar el sobreentrenamiento de los datos o porque existen restricciones de tiempo.

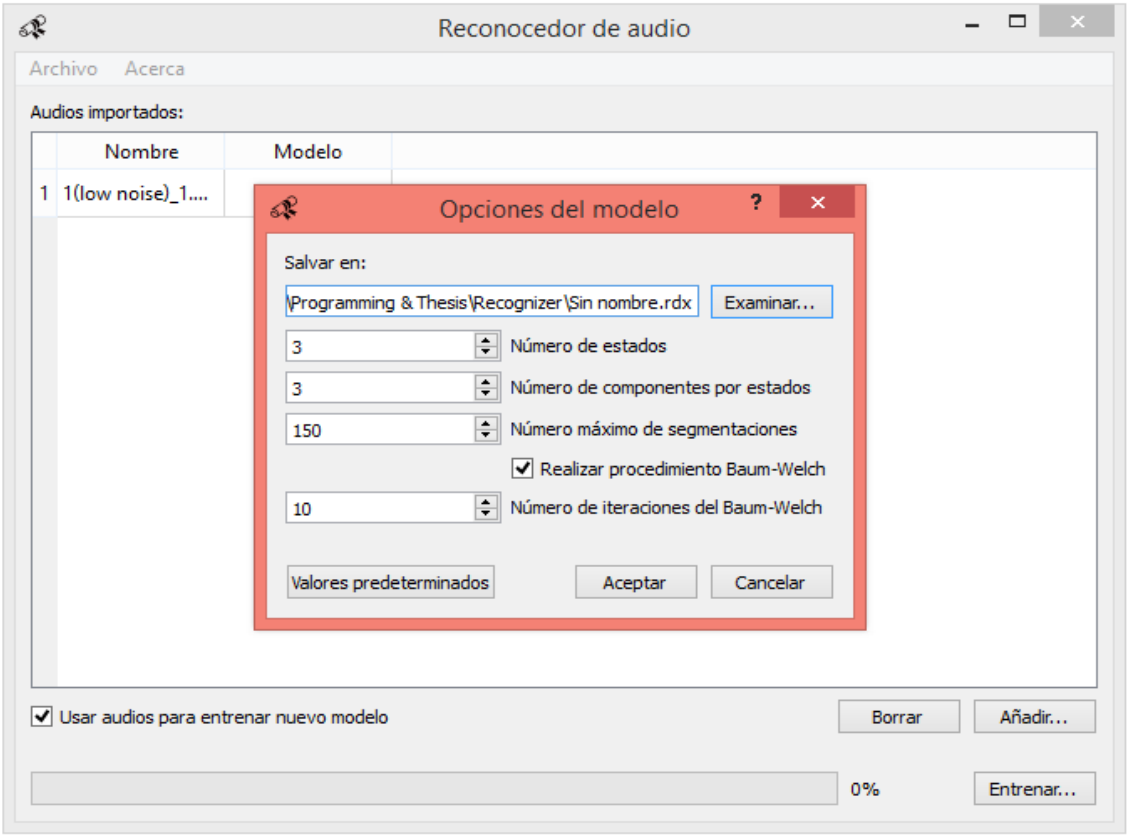

Fig. 17 Ilustración de las opciones disponibles para la creación de un nuevo modelo.

binding.py

Este módulo es el encargado de manejar el enlace entre la capa de lógica y la interfaz. En nuestro caso entre los módulos de *logic* y *ui*. En él se define la clase *AppState* que se utiliza para almacenar la información que proviene de la interfaz gráfica y pasarlo a la capa de procesamiento. También se implementan las funciones *train* y *recognize* las cuales crean los objetos *TrainModel* y *RecognizeModel* con los parámetros dados por el usuario y ejecutan estas tareas en hilos separados hasta su terminación.

Capítulo 4: Resultados

Para probar el rendimiento del ASR hemos empleado dos bases de datos: B1, que está compuesta de 12.000 archivos de audio pertenecientes a 12 especies de murciélagos, las cuales poseen 1.000 ejemplares cada una; y B2 que está compuesta de 3.253 archivos repartidos en 4 especies de aves que, a diferencia de B1 poseen cantidades diferentes cada una.

Información sobre la Base de Datos #1

Los audios en esta base de datos se encuentran a una frecuencia de muestreo de 250.000 Hz, debido a que los murciélagos emiten la mayor parte de la información en el rango ultrasónico. Además los archivos no son grabaciones extensas del animal, sino pequeños segmentos con el sonido de interés. La duración promedio para estos es de 10ms, y debido a la alta frecuencia de muestreo esta longitud es suficiente para obtener una cantidad representativa de muestras.

Los archivos de B1 han sido generados utilizando funciones matemáticas que describen la "forma" del espectrograma característico a cada una de las especies de murciélagos. Luego, modificando aleatoriamente parámetros de la señal como: duración, frecuencia máxima y mínima, intensidad de la fundamental, armónicos, etc. Se obtienen instancias reconocibles de la especie, pero que a su vez presentan variaciones únicas que simulan a un individuo específico de esta. La Fig. 18 muestra cómo se relacionan la señal real, con varias muestra aleatorias de esta.

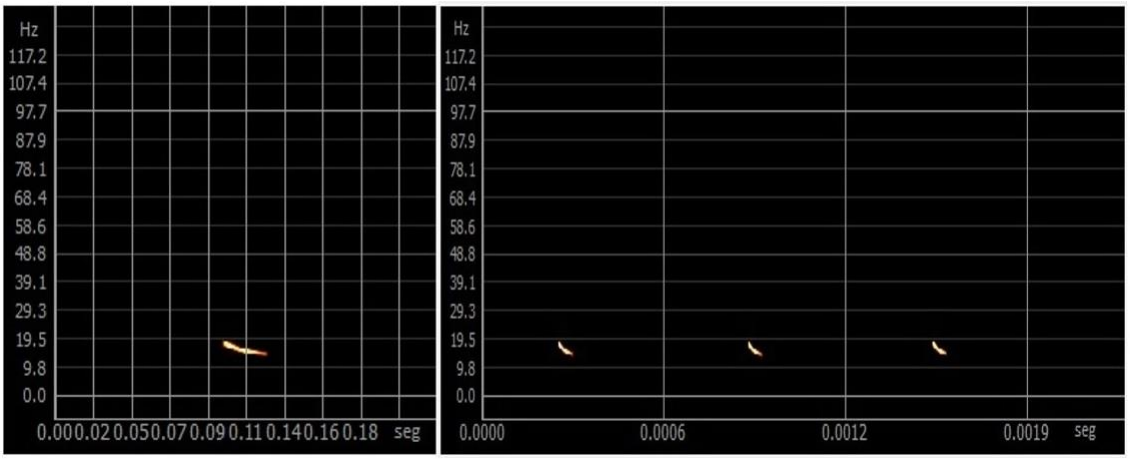

Fig.18 A la izquierda, el espectrograma de la señal real de un murciélago de la especie 1. A la derecha, tres instancias aleatorias generadas a partir de esta.

Un punto a favor en el empleo de esta base de datos es que cada archivo generado corresponde a individuos diferentes de la especie, lo cual crea una gran diversidad en los datos. Además, podemos estar seguros de que es específicamente la especie que nos interesa gracias a la ausencia de ruido y a que podemos controlar cada parámetro de su creación.

Sin embargo, esta ausencia de ruido también provoca un problema. Pues en situaciones prácticas es muy improbable que se utilicen archivos de audio completamente limpios, por lo que estaríamos obviando cómo afectaría al rendimiento del modelo la presencia de ruido en los segmentos de entrenamiento y testeo. Es por esta razón que también hemos empleado la base de datos B2.

Información sobre la Base de Datos #2

Esta base de datos ha sido obtenida de [24] y, originalmente, se encuentra formada por miles de archivos de audio pertenecientes a distintos tipos de aves. De ella se han extraído los audios pertenecientes a solo 4 especies y se han segmentado sus archivos usando un algoritmo que extrae todos aquellos segmentos de la señal que posean frecuencias con intensidades superiores a un umbral fijo.

Como resultado, y debido a que estos archivos de audio son grabaciones reales, muchos segmentos extraídos serán de ruido o de otros animales que se encontraban en el momento de la grabación. Por lo que a diferencia de con B1, en esta ocasión el HMM tendrá que entrenarse con ruido incluido. Lo cual nos permitirá evaluar su rendimiento en estas condiciones.

Además, también se ha procurado que el número de segmentos varíe de especie en especie, permitiéndonos analizar cómo diferentes tamaños para el grupo de entrenamiento afecta el entrenamiento de los modelos.

Resultados de la Base de Datos #1

En este conjunto de datos se tienen 12 especies de murciélagos que denotaremos como $M1, M2, \dots, M12$. Lo que se desea validar con las pruebas sobre B1 es que:

- Los parámetros internos del modelo estén siendo correctamente entrenados para cada especie.

Por tanto, como cada especie cuenta con exactamente 1.000 segmentos de audio, se emplearán conjuntos de entrenamiento y validación de igual tamaño, así como igual número de estados y componentes en los modelos (1 estado y 3 componentes). Garantizando que las condiciones iniciales para los reconocedores de cada especie sean similares, por lo que los resultados reflejarán más precisamente si los cambios provocados durante el entrenamiento fueron positivos.

En la primera prueba realizaremos un *2-fold cross-validation* sobre los datos de cada especie, por tanto, se dividirá el conjunto de los segmentos en dos grupos: $G1$ y $G2$. Luego, cada grupo será empleado en el entrenamiento de los modelos, mientras el otro se utilizará para la validación.

Antes de mostrar los resultados es importante aclarar que se emplearán matrices de confusión para su anotación. En estas las columnas corresponden a la respuesta esperada para el conjunto de datos que se esté validando, mientras que las filas representan todas las opciones de respuesta que el sistema puede dar. Por tanto, el valor de la intersección (a, b) con $a \neq b$ representaría el número de veces que el sistema respondió a cuando debió responder b, por lo que el valor de (b, b) es el número de respuestas acertadas. También se incluye una columna extra que muestra el porciento de respuestas correctas para cada especie, así como una fila extra donde se ubican el número total de pruebas hechas para esa columna. Por último, en la intersección entre la columna y la fila extra se ubica el porciento de correctitud global.

Entrenados con G1		**Respuestas esperadas**												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	500	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	500	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	500	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	500	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	500	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	500	0	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	500	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	500	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	500	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	500	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	500	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	500	***100%***
	Total:	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***100%***

Entrenados con G2		**Respuestas esperadas**												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	500	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	500	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	500	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	500	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	500	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	500	0	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	500	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	500	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	500	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	500	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	500	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	500	***100%***
	Total:	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***500***	***100%***

Los resultados anteriores reflejan un rendimiento perfecto del ASR para cada una de las especies de murciélagos. Sin embargo, debemos tener en cuenta que hemos estado trabajando con los datos de B1 y que además hemos empleado la mitad de los segmentos disponibles para cada especie durante el entrenamiento. Por tanto, qué pasaría si se recortase el número de archivos usados en el entrenamiento a solo un 10% de los que se emplearon en la validación anterior.

Considérese ahora un *20-fold cross-validation*, pero en vez de emplear 19 grupos para el entrenamiento y solo uno para la validación, emplearemos solo 1 grupo para el entrenamiento y 19 para la validación. Luego, como cada especie posee 1.000 segmentos, 50 de estos se emplearán para el entrenamiento, mientras que los 950 restantes servirán para validar.

En este caso los grupos en que se dividirán los segmentos de cada especie son $G1, G2, \dots, G20$. Por razones de espacio omitiremos las tablas en que se obtengan resultados perfectos.

Entrenados con G1		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	949	0	0	0	0	0	0	0	0	***99,89%***
	M5	0	0	0	0	949	0	0	0	0	0	0	0	***99,89%***
	M6	0	0	0	1	1	950	0	4	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	946	0	0	0	0	***99,57%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,94%***

Entrenados con G3		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	950	0	0	0	0	0	3	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	947	***99,68%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,97%***

Entrenados con G4		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	943	0	0	0	0	0	0	0	0	***99,26%***
	M5	0	0	0	0	936	0	0	0	0	0	0	0	***98,52%***
	M6	0	0	0	7	14	950	0	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,81%***

Entrenados con G6		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	912	0	0	0	0	0	0	0	0	***96%***
	M5	0	0	0	0	949	0	0	0	0	0	0	0	***99,89%***
	M6	0	0	0	38	1	950	0	0	0	0	0	13	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	937	***98,63%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,54%***

Entrenados con G7		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	950	27	6	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	923	0	0	0	0	0	***97,15%***
	M8	0	0	0	0	0	0	0	944	0	0	0	0	***99,36%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,71%***

Entrenados con G8		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	946	0	0	0	0	0	0	0	0	***99,57%***
	M5	0	0	0	0	948	0	0	0	0	0	0	0	***99,78%***
	M6	0	0	0	4	2	950	0	1	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	949	0	0	0	0	***99,89%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,93%***

Entrenados con G9		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	937	0	0	0	0	0	0	0	0	***98,63%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	13	0	950	0	0	0	0	7	0	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	943	0	***99,26%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,82%***

Entrenados con G10		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	948	0	0	0	0	0	0	0	***99,78%***
	M6	0	0	0	0	2	950	1	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	949	0	0	0	0	0	***99,89%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,97%***

Entrenados con G12		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	936	0	0	0	0	0	0	0	0	***98,52%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	14	0	950	5	0	0	5	0	0	***100%***
	M7	0	0	0	0	0	0	945	0	0	0	0	0	***99,47%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	945	0	0	***99,47%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,78%***

Entrenados con G14		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	5	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	945	0	0	0	0	0	0	0	0	0	0	***99,47%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	949	0	0	0	0	0	0	0	0	***99,89%***
	M5	0	0	0	0	949	0	0	0	0	0	0	0	***99,89%***
	M6	0	0	0	1	1	950	6	4	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	944	0	0	0	0	0	***99,36%***
	M8	0	0	0	0	0	0	0	946	0	0	0	0	***99,57%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,85%***

Entrenados con G16		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	947	0	0	0	0	0	0	0	0	***99,68%***
	M5	0	0	0	0	949	0	0	0	0	0	0	0	***99,89%***
	M6	0	0	0	3	1	950	0	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,96%***

Entrenados con G17		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	950	2	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	948	0	0	0	0	0	***99,78%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,98%***

Entrenados con G18		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	950	2	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	948	0	0	0	0	0	***99,78%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,98%***

Entrenados con G19		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	949	0	0	0	0	0	0	0	***99,89%***
	M6	0	0	0	0	1	950	2	0	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	948	0	0	0	0	0	***99,78%***
	M8	0	0	0	0	0	0	0	950	0	0	0	0	***100%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,97%***

Entrenados con G20		Respuestas esperadas												
		M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	%
Opciones	M1	950	0	0	0	0	0	0	0	0	0	0	0	***100%***
	M2	0	950	0	0	0	0	0	0	0	0	0	0	***100%***
	M3	0	0	950	0	0	0	0	0	0	0	0	0	***100%***
	M4	0	0	0	950	0	0	0	0	0	0	0	0	***100%***
	M5	0	0	0	0	950	0	0	0	0	0	0	0	***100%***
	M6	0	0	0	0	0	950	0	1	0	0	0	0	***100%***
	M7	0	0	0	0	0	0	950	0	0	0	0	0	***100%***
	M8	0	0	0	0	0	0	0	949	0	0	0	0	***99,89%***
	M9	0	0	0	0	0	0	0	0	950	0	0	0	***100%***
	M10	0	0	0	0	0	0	0	0	0	950	0	0	***100%***
	M11	0	0	0	0	0	0	0	0	0	0	950	0	***100%***
	M12	0	0	0	0	0	0	0	0	0	0	0	950	***100%***
	Total:	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***950***	***99,99%***

Estos resultados evidencian que el número de segmentos usados en el entrenamiento afecta el rendimiento del reconocedor. Además de que para sonidos limpios el proceso de adaptación del modelo es capaz de alcanzar altas tasas de acierto, incluso cuando se emplean pequeños conjuntos para el entrenamiento.

Resultados de la Base de Datos #2

A diferencia que en B1, las pruebas sobre B2 no buscan validar el proceso de entrenamiento, sino ponerlo a prueba en diferentes situaciones típicas y cuantificar su rendimiento. Las incógnitas que se buscan esclarecer en las diferentes pruebas son:

- El papel del número de estados y componentes en el rendimiento del modelo.
- La importancia de la presencia de ruido en los datos y cómo afecta al rendimiento.

Para realizar las validaciones se eligieron de B2 cuatro especies de aves diferentes:

- *Accipiter-nisus* (**AN**): Posee 556 segmentos de audio.
- *Alopochen-aegyptiaca* (**AA**): Posee 1356 segmentos de audio.
- *Anas-platyrhynchos* (**AP**): Posee 412 segmentos de audio.
- *Branta-canadensis* (**BC**): Posee 929 segmentos de audio.

Al igual que antes la primera prueba es un *2-fold cross-validation* dividido en los grupos: $G1$ y $G2$. Inicialmente se realizará la validación usando una cantidad fija de estados y componentes (3 y 3) para todos los reconocedores. Luego, empleando el mismo par de grupos, se realizará nuevamente el *2-fold cross-validation*, pero en esta ocasión se emplearán modelos con diferentes configuraciones para cada especie. Veamos los resultados para el caso donde todos los reconocedores posean 3 estados y 3 componentes.

Entrenados con G1		**Respuestas esperadas**				
		AN33[17]	AA33	AP33	BC33	%
Opciones	AN33	195	21	2	0	***70,14%***
	AA33	4	562	1	45	***82,89%***
	AP33	55	60	203	8	***98,54%***
	BC33	24	35	0	412	***88,60%***
	Total:	***278***	***678***	***206***	***465***	***84,32%***

Entrenados con G2		**Respuestas esperadas**				
		AN33	AA33	AP33	BC33	%
Opciones	AN33	277	11	33	24	***99,64%***
	AA33	1	497	50	44	***73,30%***
	AP33	0	1	114	29	***55,33%***
	BC33	0	169	9	367	***79,09%***
	Total:	***278***	***678***	***206***	***464***	***77,18%***

Siendo el promedio total de respuestas correctas: **80,75%**.

[17] Los dígitos después del nombre de la especie representan, en ese orden, el número de estados y componentes empleadas.

Empleemos ahora configuraciones específicas en cada reconocedor y veamos si esto mejora los resultados.

- *Accipiter-nisus* (**AN**): 7 estados y 5 componentes.
- *Alopochen-aegyptiaca* (**AA**): 3 estados y 5 componentes.
- *Anas-platyrhynchos* (**AP**): 3 estados y 3 componentes.
- *Branta-canadensis* (**BC**): 7 estados y 3 componentes.

Los resultados fueron los siguientes.

Entrenados con G1		**Respuestas esperadas**				
		AN75	AA35	AP33	BC73	%
Opciones	AN75	194	13	2	0	***69,78%***
	AA35	5	566	0	41	***83,48%***
	AP33	75	64	196	7	***95,14%***
	BC73	4	35	8	417	***89,67%***
	Total:	***278***	***678***	***206***	***465***	***84,38%***

Entrenados con G2		**Respuestas esperadas**				
		AN75	AA35	AP33	BC73	%
Opciones	AN75	276	7	30	25	***99,28%***
	AA35	2	549	37	56	***80,97%***
	AP33	0	15	128	36	***62,13%***
	BC73	0	107	11	347	***74,78%***
	Total:	***278***	***678***	***206***	***464***	***79,95%***

Con un promedio total de respuestas acertadas de: **82,17%**. Lo cual es un incremento de **1,42%** con respecto a aquellos con configuraciones iguales. Indicando que tanto el número de estados como el de componentes afectan el rendimiento del ASR. Veamos ahora si aumentando el tamaño de los conjuntos de entrenamiento conseguimos incrementar el promedio de respuestas correctas.

Para eso considérese un *4-fold cross-validation*, donde se divide al conjunto de datos de cada especie en 4 grupos: $G1, G2, G3$ y $G4$. Luego se utilizan 3 de ellos para el entrenamiento y solo uno en la validación. Incrementando, por una parte, el tamaño del conjunto de entrenamiento, mientras que se disminuye el de validación.

Entrenados con G1, G2, G3		**Respuestas esperadas**				
		AN75	AA35	AP33	BC73	%
Opciones	AN75	121	13	0	0	***87,05%***
	AA35	1	297	0	38	***87,61%***
	AP33	16	8	103	8	***100%***
	BC73	1	21	0	187	***80,25%***
	Total:	***139***	***339***	***103***	***233***	***86,97%***

Entrenados con G1, G2, G4		Respuestas esperadas				
		AN75	AA35	AP33	BC73	%
Opciones	AN75	120	6	2	0	***86,33%***
	AA35	1	278	0	4	***82%***
	AP33	18	47	101	0	***98,05%***
	BC73	0	8	0	228	***98,27%***
	Total:	***139***	***339***	***103***	***232***	***89,42%***

Entrenados con G1, G3, G4		Respuestas esperadas				
		AN75	AA35	AP33	BC73	%
Opciones	AN75	139	4	15	22	***100%***
	AA35	0	307	13	6	***90,56%***
	AP33	0	2	35	1	***33,98%***
	BC73	0	26	40	203	***87,5%***
	Total:	***139***	***339***	***103***	***232***	***84,13%***

Entrenados con G2, G3, G4		Respuestas esperadas				
		AN75	AA35	AP33	BC73	%
Opciones	AN75	138	6	24	4	***99,28%***
	AA35	1	235	5	29	***69,32%***
	AP33	0	14	73	46	***70,87%***
	BC73	0	84	1	153	***65,94%***
	Total:	***139***	***339***	***103***	***232***	***73,67%***

Con un promedio total de respuestas correctas: **83,55%**, lográndose un incremento del **1,38%** con respecto al caso anterior donde los modelos poseían las mismas configuraciones, pero usaban conjuntos de entrenamiento más pequeños. No obstante, a pesar de esto no logramos igualar aún las tasas de detección alcanzadas en B1 con el *20-fold inverse cross-validation*[18]. Lo cual evidencia la degradación en el rendimiento de los reconocedores ante la presencia de señales con ruido.

Luego, en estas pruebas podemos constatar la potencia de los HMMs como reconocedores de audios de animales, pero también nos encontramos con que existen numerosos factores que pueden afectar dicho rendimiento como la calidad de los archivos de audio, la configuración elegida para los modelos y/o el número de ejemplares utilizados para el entrenamiento de estos.

[18] Lo de *inverse* se refiere a que utilizábamos 1 grupo para entrenar y 19 para validar, en vez de 19 para el entrenamiento y solo 1 para la validación.

Conclusiones

Los Modelos Ocultos de Markov son adecuados para ser utilizados como reconocedores automáticos de sonidos de animales y su empleo en la construcción de sistemas como el mostrado, puede resultar de gran utilidad en el estudio de poblaciones de animales. De estas podríamos determinar de forma automática y remota informaciones como su cantidad, costumbres migratorias y alimenticias, épocas de apareamiento, tipos de articulaciones sonoras y cambios en su comportamiento basados en la voz.

Además, debido a que los modelos te permiten elegir el número de estados, componentes, así como las funciones de densidad y tipo de matrices de transición, es posible adaptarlos con relativa facilidad a una amplísima variedad de animales y situaciones. Incluso, con una buena implementación, se puede lograr que los usuarios finales no necesiten demasiados conocimientos técnicos relacionados con el modelo, lo cual permitiría ampliar la gama de personas a la que el producto está destinado, y por tanto, incrementar el número de escenarios en los que sería útil.

Referencias

[1] J. K. Baker, «The dragon system-An overview,» *IEEETrans,* p. 2249, 1975.

[2] F. Jelinek, «Continuous speech recognition by statistical,» *IEEE,* pp. 532-536, 1976.

[3] L. E. Baum y T. Petrie, «Statistical inference for probabilistic functions of finite state Markov chains,» vol. 37, pp. 1554-1563, 1966.

[4] S. Shaw, «An Evaluation of Birdsong Recognition Techniques,» p. 69, 2011.

[5] D. J. Ross, Bird Call Recognition with Atificial Neural Networks, Support Vector Machines and Kernel Density Estimation, Winnipeg, 2006.

[6] D. Reynolds, «Gaussian Mixtures Models,» p. 5.

[7] I. V. M. Ruiz, «Golem,» 28 03 2013. [En línea]. Available: http://golem.iimas.unam.mx/%7Eivanvladimir/es/.

[8] Y. Ren, M. T. Johnson, P. J. Clemins, M. Darre, S. S. Glaeser, T. S. Osiejuk y E. Out-Nyarko, «A Framework for Bioacoustic Vocalization Analysis Using Hidden Markov Models,» *Algorithms,* pp. 1410-1428, 2009.

[9] V. M. Trifa, A. N. G. Kirschel, C. E. Taylor y E. E. Vallejo, «Automated species recognition of antbirds in a Mexican rainforest using hidden Markov models,» *AIP,* p. 9, 2007.

[10] I. D. Agranat, «Automatically identifying animal species from their vocalisations,» 2009.

[11] P. Mermelstein, «Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences,» *IEEE Transactions on Acoustics, Speech, and Signal Processing,* vol. 28, nº 4, pp. 357-366, 1980.

[12] J. Feldman, «The Fourier Transform,» p. 14, 2007.

[13] J. Lyon, «Practical Cryptography,» 2012. [En línea]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

[14] «Wikipedia,» 2013. [En línea]. Available: es.wikipedia.com\Ventana (funcion).

[15] L. R. Rabiner, «A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,» *IEEE,* pp. 1-30, 1989.

[16] L. E. Baum y J. A. Egon, «An inequality with applications to statistical estimation for probabilistic functions of a Markov process and to a model for ecology,» *Bull. Amer. Meterol. Soc.,* vol. 73, pp. 360-363, 1967.

[17] L. E. Baum y G. R. Sell, «Growth functions for transformations on manifolds,» *Pac. J. Math.,* vol. 27, nº 2, pp. 211-227, 1968.

[18] A. J. Viterbi, «Error bounds for convolutional codes and an asymptotically optimal decoding algorithm,» *IEEETrans. Informat Theory,* Vols. %1 de %2IT-13, pp. 260-269, 1967.

[19] G. D. Forney, «The Viterbi algorithm,» *Proc. IEEE,* vol. 61, pp. 268-278, 1973.

[20] L. A. Liporace, «Maximum likelihood estimation for multivariate observations of Markov sources,» *IEEETrans. Informat. Theory,* Vols. %1 de %2IT-28, nº 5, pp. 729-734, 1982.

[21] B. H. Juang, S. E. Levinson y M. M. Sondhi, «Maximum likelihood estimation for multivariate mixture observations of Markov chains,» *IEEETrans. Informat. Theory,* Vols. %1 de %2IT-32, nº 2, pp. 307-309, 1986.

[22] S. E. Levinson, L. R. Rabiner y M. M. Sondhi, «"An introduction to the application of the theory of probabilistic functions of a Markov processto automatic speech recognition,» *Bell Syst. Tech. J.,* vol. 62, nº 4, pp. 1035-1074, 1983.

[23] A.P.Dempster, N. Laird y D. Rubin, «Maximum lilelihood from incomplete data via EM algorithm,» *J. Roy. Stat Soc,* vol. 30, nº 1, pp. 1-38, 1977.

[24] [En línea]. Available: http://www.xenox.org. [Último acceso: Mayo 2015].