UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Proyecto Final
2016

Reconocimiento Embebido de Habla Aislada Independiente del Hablante en Tiempo Real con Modelos Ocultos de Markov y Máquinas de Vectores de Soporte

Tesis de grado

Mariano Marufo da Silva

Tesis de grado

Título: Reconocimiento Embebido de Habla Aislada Independiente del Hablante en Tiempo Real con Modelos Ocultos de Markov y Máquinas de Vectores de Soporte

Alumno: Marufo da Silva, Mariano (mmarufodasilva@est.frba.utn.edu.ar)

Director: Evin, Diego Alexis (Laboratorio de Investigaciones Sensoriales-INIGEM, CONICET-UBA)

Tutor UTN: Verrastro, Sebastian

Asignatura: Proyecto Final

Carrera: Ingeniería Electrónica

Universidad: Universidad Tecnológica Nacional - Facultad Regional Buenos Aires

Año: 2016

Resumen

El estudio de algoritmos de reconocimiento automático del habla como un problema dentro del área de la inteligencia artificial tiene ya varias décadas. Sin embargo, recién en los últimos años el advenimiento de nuevas tecnologías que brindan una capacidad adicional de interacción usuario-máquina a través de la voz, sumado a las masivas bases de datos que poseen las grandes empresas de internet, han hecho que hoy en día sea algo común para muchas personas realizar tareas como una búsqueda de información en la web o dar una instrucción a su teléfono celular por medio del habla.

Si bien los algoritmos de reconocimiento automático del habla en plataformas embebidas han alcanzado niveles de rendimiento consistentes con requerimientos de aplicaciones comerciales, en la mayoría de los casos este rendimiento es alcanzado a expensas de transferir la complejidad del problema a servidores remotos. Es decir, se comprime la señal de voz en el dispositivo utilizado, se envía la información comprimida a servidores (posiblemente a miles de kilómetros de distancia) y recién allí se realiza la tarea de reconocimiento en sí misma [**1**]. Para muchas aplicaciones esto representa un problema ya que en primer lugar requiere necesariamente de una conexión estable a internet y en segundo lugar la velocidad del reconocimiento estará fuertemente limitada por el ancho de banda disponible. Por otro lado, este procedimiento compromete la privacidad del usuario que utiliza dicho servicio. La solución propuesta es la de implementar el algoritmo de reconocimiento directamente en el dispositivo utilizado (por ejemplo, un celular o la computadora de abordo de un automóvil).

Inicialmente se utilizaron técnicas de Dynamic Time Warping (DTW), que con el paso del tiempo fueron reemplazadas por métodos de modelización del habla basados en Modelos Ocultos de Markov (HMMs) y utilizando el algoritmo de decodificación de Viterbi para realizar el reconocimiento. En los años 1980 se comenzó a investigar en métodos que reemplazaran a este algoritmo por Redes Neuronales Artificiales (ANNs), y posteriormente, en los años 1990, por Máquinas de Vectores de Soporte (SVMs), obteniendo aún al día de hoy resultados exitosos [**2**]. Si bien tanto ANN como SVM

son métodos del estado del arte en reconocimiento de patrones, por sí mismos no pueden modelar eficientemente las variaciones temporales del habla. Es por eso que lo que se utilizan son en realidad sistemas híbridos que combinen el poder de reconocimiento de los mismos con la capacidad de modelización del habla de los HMMs, existiendo entonces sistemas híbridos HMM/ANN o HMM/SVM.

Como se pretende utilizar un procesador embebido, es preferible limitar el vocabulario a reconocer ya que los requisitos de procesamiento aumentarían con el tamaño del mismo. Por otro lado, ANNs servirían para un análisis de escritorio, pero su principal problema es que suelen ser muy sensibles al ruido y a variaciones de los canales, lo cual genera muchas complicaciones con mínimos locales al querer adaptarlas a reconocimiento del habla. Si se pretende hacer un desarrollo en software embebido que trabaje en tiempo real, SVMs son más convenientes que ANNs, en especial si se tiene una limitada base de datos. Además, SVMs son muy robustas frente al ruido, lo cual ayuda a combatir el *overfitting* (sobreajuste) y así tener mayor probabilidad de generalizar mejor (característica fundamental de un reconocedor de patrones) [**3**].

En este trabajo se propone entonces realizar el diseño de un algoritmo reconocedor de habla en tiempo real utilizando HMMs y SVMs como algoritmo de reconocimiento para su utilización en un sistema basado en palabras aisladas, de vocabulario pequeño (no más de algunas decenas de palabras) y para condiciones de uso en laboratorio (buena relación señal a ruido).

En general los algoritmos de reconocimiento de patrones suelen tener primero una etapa de entrenamiento en la cual "aprenden" a partir de información conocida, una etapa de evaluación para estimar cuán bien aprendieron y luego la etapa de reconocimiento propiamente dicho (es decir, aplicación del algoritmo).

Se pretende realizar la etapa de entrenamiento del algoritmo (obtención de los modelos que representan a las palabras a reconocer) y la evaluación del mismo de forma offline en una PC utilizando una base de datos acorde a las características del reconocedor. Por otro lado, se desea implementar la aplicación del algoritmo como un reconocedor propiamente dicho en tiempo real en una placa de desarrollo basada en un microcontrolador de 32-bit ARM Cortex-M4F y comparar sus resultados con los de un trabajo de investigación basado en un algoritmo tradicional de reconocimiento del habla implementado en la misma plataforma.

Agradecimientos

Quisiera dedicar unas palabras para agradecer a las personas que de diferentes maneras me apoyaron y ayudaron a que pueda llevar a cabo esta tesis de investigación como trabajo final de mi carrera universitaria.

En primer lugar, estaré eternamente agradecido a Diego Evin, quien desde el momento en el que aceptó ser mi director de tesis, no dejó de colaborar con su enorme experiencia y conocimiento teórico y práctico de temas que yo al comienzo no manejaba en profundidad, pero sobre todo con su tiempo e interminable predisposición a ayudarme a despejar dudas, darme ideas de cómo resolver problemas, revisiones de la documentación, etc., ya sea por mail o en persona en su laboratorio, y en días de semana, fines de semana o incluso sus vacaciones.

También quisiera agradecer a los miembros del LIS (Laboratorio de Investigaciones Sensoriales), uno de los lugares de trabajo de Diego, por colaborar amablemente donando su tiempo (y su voz) en la grabación de los audios para la confección de una de las bases de datos utilizadas en este trabajo, y por permitirme para tal fin hacer uso de la cámara anecoica que poseen en sus instalaciones.

Por parte de la universidad, les agradezco a Sebastian Verrastro y la cátedra de Proyecto Final, quienes me dieron libertad para realizar una tesis de investigación en un tema no tan convencional para el proyecto final de ingeniería electrónica, y realizaron revisiones y aportes muy valiosos a medida que les fui presentando avances a lo largo de la tesis. Por otro lado, también tengo que destacar la colaboración de Alejandro Alvarez, quien realizó previamente una tesis relacionada con el mismo tema y me dejó reutilizar su arquitectura implementada, lo cual me permitió enfocarme en el desarrollo de los algoritmos.

Por último, agradezco a mi mamá Rosa, mi papá Raúl, mi hermano Maxi y mis amigos, por el apoyo y la paciencia que tuvieron conmigo no sólo a lo largo de esta tesis, sino por todo lo que hicieron por mí a lo largo de toda la carrera y, por qué no, de mi vida.

Índice

Índice de figuras

Índice de cuadros

1. Reconocimiento Automático del Habla

1.1. Introducción

Un sistema de reconocimiento automático del habla típicamente se compone de 2 módulos principales:

- Módulo de procesamiento de la señal de voz:

  Generalmente comienza con un transductor (micrófono) que convierte la señal de voz en una señal eléctrica, la cual luego se digitaliza. Luego siguen etapas de DSP que efectúan diferentes procesamientos a la misma según la aplicación deseada, por ejemplo: separación de la voz del silencio dentro de la señal, segmentación de la voz en pequeños intervalos para procesarlos de forma independiente, transformación de los segmentos de voz al dominio de la frecuencia, parametrización de los segmentos para la obtención de sus características útiles para el reconocimiento, etc.

- Módulo de reconocimiento de los patrones pronunciados:

  Se comparan las características obtenidas a partir de las diferentes palabras pronunciadas con las características de los modelos correspondientes a las palabras conocidas por el sistema (su diccionario) para determinar a partir de procedimientos de reconocimiento de patrones cuáles fueron las pronunciadas por el hablante.

El siguiente es un diagrama en bloques de un sistema de reconocimiento automático del habla (ASR) completo (en esta tesis no se utilizan los N-gramas ni el Lexicón):

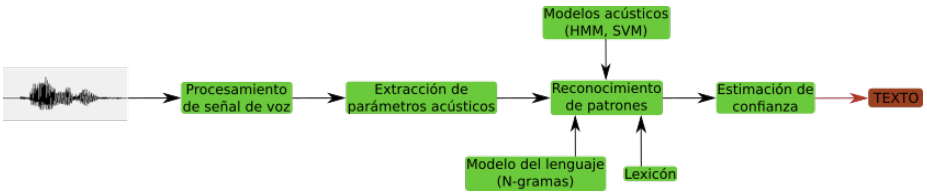

Figura 1: Diagrama en bloques de un sistema de ASR.

1.2. Extracción de parámetros acústicos

Como suele suceder en los distintos problemas relacionados con el reconocimiento de patrones, los algoritmos utilizados para el reconocimiento automático del habla no trabajan modelando directamente las señales que representan las ondas acústicas, sino que requieren previamente de un trabajo de extracción de parámetros o características de dichas señales. Esto se debe en primer lugar a que las señales temporales del habla son muy redundantes (procesando los datos en el dominio del tiempo a la frecuencia de muestreo original se tienen muchos puntos próximos con valores similares) y en segundo lugar a que construir modelos acústicos temporales resulta muy difícil y poco práctico.

Existen distintos parámetros utilizados en el procesamiento del habla en general como por ejemplo Pitch Period (período con el que vibran las cuerdas vocales), Short Time Energy (sumatoria de la energía de cada muestra de un intervalo, generalmente de entre 10 y 20 mseg) y ZCC-Short Time Zero Cross Count (se filtra la continua y

se cuentan los cruces por cero, para saber si la señal es de alta o baja frecuencia en ese intervalo). Sin embargo, particularmente en lo que respecta al reconocimiento del habla se suelen utilizar parámetros espectrales, siendo los más utilizados los llamados MFCCs (Mel-frequency cepstral coefficients).

Para la obtención de los coeficientes MFCC, se siguen los siguientes pasos a partir de la señal acústica temporal [**4**]:

- En lo que respecta al reconocimiento de palabras aisladas, el hecho de separar el ruido de fondo (de baja energía en el caso de uso de laboratorio) de la señal de voz aporta mejoras significativas al rendimiento del reconocedor. Una manera simple de implementar esto, que tiene una buena respuesta cuando se habla con bajo ruido de fondo es utilizando técnicas que diferencien ”habla” de ”no-habla” a partir del nivel de energía de la señal durante un cierto tiempo.

- Filtro de pre-énfasis: se trata de un filtro pasa altos. Los sonidos del habla son producidos por el aire que pasa en primer lugar por las cuerdas vocales (las cuales vibran cuando se pronuncia un sonido *sonoro*[1] (voiced sound) y se mantienen abiertas cuando se pronuncia un sonido *sordo* (unvoiced sound)) y luego a través del tracto vocal (desde las cuerdas vocales hasta los labios) el cual define las características espectrales de la onda sonora según se quiera pronunciar uno u otro fonema. Aplicar un filtro pasa altos de pre-énfasis tiene la finalidad de realzar las componentes de alta frecuencia que sufren de una atenuación (conocida como *spectral tilt*) debido a las características que presenta el pulso glótico. Este filtro se caracteriza por la siguiente ecuación:

$$\widehat{x_n} = x_n - a.x_{n-1} \tag{1}$$

  Donde a suele valer aproximadamente 0.95.

- Ventaneo: de la señal completa se toman intervalos (de por ejemplo 20 a 30 mseg) con una separación tal (de por ejemplo 10 mseg) que queden solapados entre sí. Cada intervalo es luego multiplicado por una ventana (en general, de Hamming) de bordes suaves para reducir la introducción de componentes espectrales espúreos en el proceso de segmentación.

- |FFT|: se calcula el módulo de la transformada de Fourier de cada intervalo.

- Escala de Mel: se mapea el espectro en la escala de frecuencias a la escala de Mel, utilizando un banco de filtros triangulares solapados. Estos filtros imitan las diferentes sensibilidades en frecuencia del oído humano.

- Log: se calcula el logaritmo a cada uno de los filtros en la escala de Mel.

- DCT: se calcula la transformada coseno discreta a los logaritmos. A los coeficientes obtenidos en esta etapa se los llama cepstrum.

- Velocidad y aceleración: cálculo de la 1° y 2° derivada del cepstrum. Teniendo en cuenta que el cepstrum provee información estática mientras que el habla es dinámica, estas derivadas permiten modelar la dinámica de variación espectral entre *frames* sucesivos. El vector de observaciones utilizado para entrenamiento

[1] Los sonidos *sonoros* son aquellos durante los cuales las cuerdas vocales producen vibración, como las vocales, consonantes sonoras (/b/, /d/, /q/, /l/, etc.), consonantes nasales (/m/, /n/), etc.

y reconocimiento es el que surge de la concatenación del cepstrum con sus derivadas. También suele concatenarse información correspondiente a una medida de la energía del cepstrum.

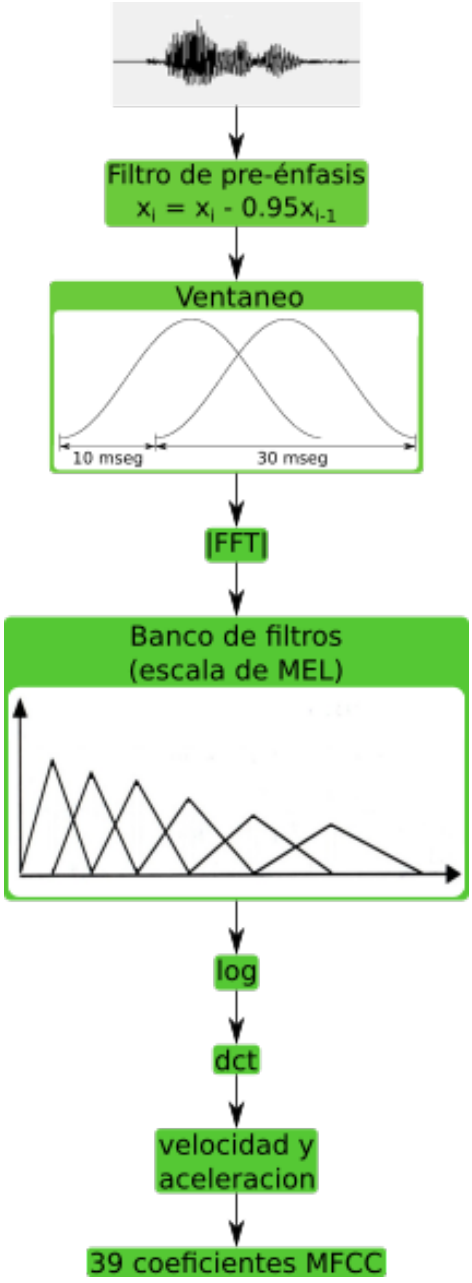

Figura 2: Pasos para la extracción de los coeficientes MFCC.

2. Modelos Ocultos de Markov: desarrollo teórico[2]

2.1. Reconocimiento de palabras aisladas

Representaremos las características de las palabras pronunciadas a partir de la secuencia de vectores de observaciones O:

$$O = o_1, \ldots, o_T \tag{2}$$

donde cada o_t es el vector de observaciones correspondiente al instante t y T es la cantidad total de vectores de observaciones. Entonces, el objetivo del reconocimiento automático de palabras aisladas es el de resolver:

$$arg \max_{1 \leq i \leq V} \left(P\left(w_i|O\right) \right) \tag{3}$$

donde V es la cantidad de palabras en el diccionario y cada w_i es la palabra número i del mismo. Aplicando el teorema de Bayes obtenemos:

$$P\left(w_i|O\right) = \frac{P\left(O|w_i\right).P\left(w_i\right)}{P\left(O\right)} \tag{4}$$

Es decir que para un set de probabilidades $P\left(w_i\right)$ iguales, la palabra más probable de haber sido pronunciada depende sólo de $P\left(O|w_i\right)$. Debido al gran rango de valores que puede tomar la secuencia O, el cálculo directo de $P\left(o_1, \ldots, o_T \ |w_i\right)$ no es posible, razón por la cual se recurre a modelar la generación de las palabras a partir de un modelo paramétrico.

La producción del habla puede representarse por un modelo estadístico compuesto por estados con transiciones entre ellos (las transiciones tienen diferentes probabilidades de acontecer, aquellas que no son posibles tienen probabilidad cero) y una función de densidad de probabilidad de emisión de distintos sonidos (por ejemplo, fonos[3]) para cada uno de los estados. De esta forma, en el ASR basado en HMM (Hidden Markov Model) cada modelo acústico se representa por un modelo oculto de Markov. En particular, en el problema de reconocimiento de palabras aisladas, cada palabra del diccionario del sistema puede tener su propio HMM que la represente o bien se puede usar también modelos de sub-palabras. Es decir, se puede tener un reconocedor basado en HMM para palabras aisladas en que cada HMM se corresponda, por ejemplo, a un monofono. En este trabajo se modelará cada palabra con un HMM. En la siguiente figura observamos un ejemplo de cómo se representa la generación de la secuencia de vectores de observaciones correspondientes a cada palabra pronunciada a partir de un HMM:

[2] La sección "Modelos Ocultos de Markov: desarrollo teórico" corresponde a un resumen teórico de los contenidos desarrollados en [**5**] y [**6**].

[3] Fonema es una entidad abstracta que comprende un infinito de realizaciones físicas que se mapean a una misma entidad lingüística, mientras que fono es cada una de esas realizaciones posibles. Así, por ejemplo, el fonema /a/ tiene infinitos sonidos o realizaciones asociados. Cada una de esas realizaciones es un fono y al conjunto de fonos que se mapean al mismo fonema se lo llama alófono.

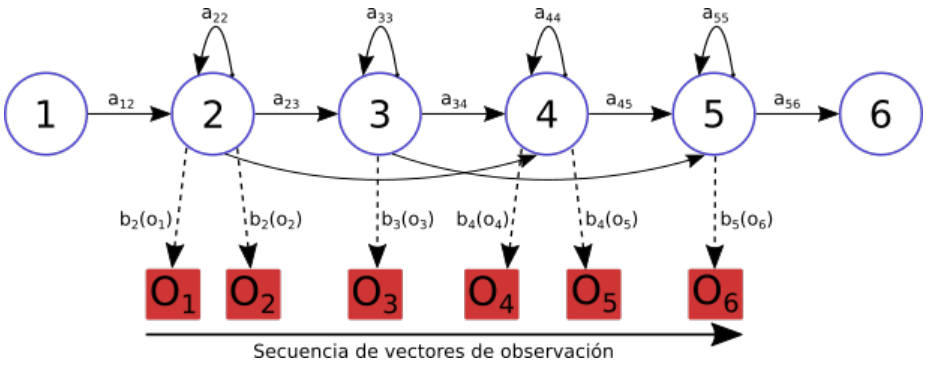

Figura 3: Ejemplo de representación de un HMM.

Un modelo de Markov es una máquina de N estados finita que en cada instante t actualiza su estado, donde la transición del estado i al estado j se da a partir de la probabilidad discreta a_{ij}. Según el estado j en el que se encuentre en cada instante t genera un nuevo vector de observaciones o_t (se lo llama con el subíndice t ya que el mismo indica el instante en el que se produjo dicho vector de observaciones) a partir de la densidad de probabilidad de salida $b_j\,(o_t)$.

Teniendo en cuenta que es un modelo probabilístico, se cumplen las siguientes condiciones:

- Al tratarse de una cadena de Markov, la dependencia probabilística es truncada solamente al estado actual y al anterior, es decir (llamaremos q_t al estado en el que se encuentra el modelo en el instante t):

$$P\,(q_t = S_j | q_{t-1} = S_i,\ q_{t-2} = S_k,\ \ldots) = P\,(q_t = S_j | q_{t-1} = S_i) \tag{5}$$

- $B = \{b_j\,(o_t)\}$ es la matriz de probabilidades de emisión de observaciones, donde $b_j\,(o_t)$ es la probabilidad de emisión de la observación o_t si el modelo se encuentra en el estado S_j en el instante t.

- $A = \{a_{ij}\}$ es la matriz de probabilidades de transiciones de estados, donde se cumple:

$$a_{ij} = P\,(q_{t+1} = S_j | q_t = S_i) \text{ para } 1 \leq i \text{ y } j \leq N \tag{6}$$

$$a_{ij} \geq 0 \quad \forall i, j \tag{7}$$

$$\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i \tag{8}$$

- $\pi = \{\pi_i\}$ es la matriz de probabilidades de estados iniciales, donde se cumple:

$$\pi_i = P\,(q_1 = S_i) \text{ para } 1 \leq i \leq N \tag{9}$$

Como vemos, un modelo oculto de Markov es un proceso doblemente estocástico donde el proceso que genera la secuencia de estados no es observable (está oculto) mientras que el único proceso observable es el que genera la secuencia de vectores de observaciones O. Una descripción completa de un HMM requiere la especificación del parámetro N (cantidad de estados del modelo) así como también de las matrices de probabilidades A, B y π, para las cuales se suele utilizar la nomenclatura $\lambda = (A, B, \pi)$.

En el reconocimiento automático de palabras aisladas se tiene un conjunto de modelos λ_i (obtenidos a través de un proceso de entrenamiento) que representan las w_i palabras del diccionario. Entonces, para resolver la ecuación $arg \max_{1 \leq i \leq V} (P(w_i|O))$ (a través del teorema de Bayes) asumimos lo siguiente:

$$P(O|\lambda_i) = P(O|w_i) \tag{10}$$

La utilización de HMMs para el reconocimiento automático de palabras aisladas consta de dos etapas, representadas en las siguientes figuras:

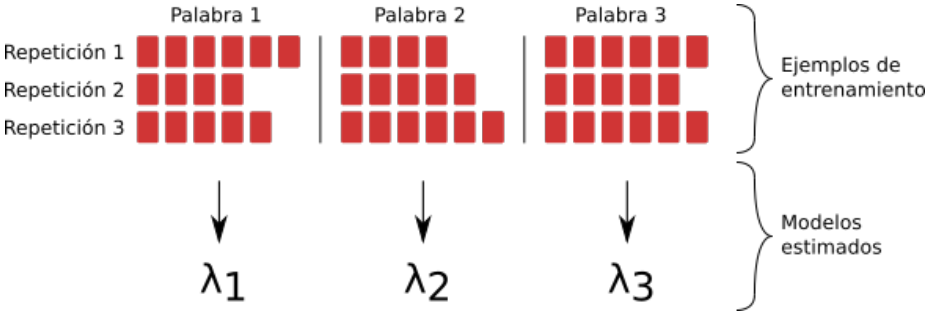

Figura 4: Etapa de entrenamiento.

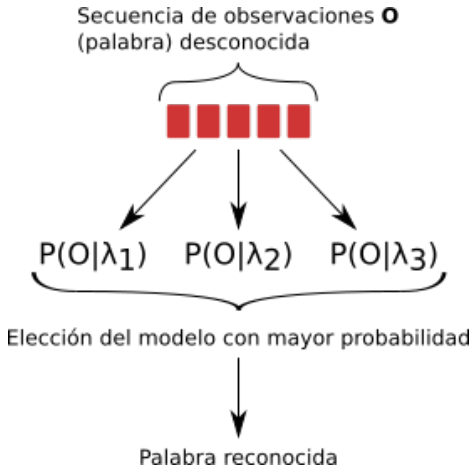

Figura 5: Etapa de testeo/reconocimiento.

- Etapa de entrenamiento: se tiene un set de entrenamiento compuesto por varias repeticiones (distintas de secuencias de vectores de observaciones) para cada una de las V palabras del diccionario. Para cada palabra del diccionario del sistema se debe construir un modelo λ_i que la represente, es decir, se debe estimar los parámetros A, B y π del modelo que maximicen la verosimilitud del mismo con la palabra que representa a partir de las repeticiones disponibles de secuencias de vectores de observaciones correspondientes a cada palabra w_i. Esta etapa es llevada a cabo a partir de un eficiente procedimiento de reestimación de parámetros llamado algoritmo de Baum–Welch. De esta manera, considerando que se posee un set de entrenamiento con la suficiente cantidad de repeticiones para cada palabra, se puede construir un HMM para cada una de ellas que modele implícitamente las diferentes fuentes variables del habla real.

- Etapa de testeo/reconocimiento: para cada palabra desconocida que se pretende reconocer se debe calcular la probabilidad de cada modelo de generar dicha secuencia de observaciones y luego se elige el modelo más probable.

Entonces, el proceso de reconocimiento propiamente dicho consiste en obtener la secuencia de vectores de observaciones O (donde las observaciones son alguna representación de ciertas características, en nuestro caso utilizamos los coeficientes MFCC) de la señal de voz correspondiente a la palabra pronunciada, seguido por un cálculo de las verosimilitudes para cada uno de los modelos $P(O|\lambda_i)$ para $1 \leq i \leq V$ y finalmente de la selección de la palabra cuyo modelo posea la máxima verosimilitud: $i^* = arg \max_{1 \leq i \leq V} (P(O|\lambda_i))$. En esta etapa generalmente se utiliza un algoritmo llamado procedimiento *forward* para calcular las probabilidades, lo cual requiere una cantidad de cálculos en el orden de $V.N^2.T$ (esto no representa un problema para procesadores modernos).

2.2. Elección del tipo de HMM

El tipo de HMM que utilizaremos será el llamado modelo Bakis o izquierda-derecha (figura 6) [7]. Este es el tipo de modelo más apropiado para reconocimiento de palabras aisladas con un HMM para cada palabra del diccionario ya que las secuencias de estados asociadas con el mismo tienen la propiedad de que a medida que el tiempo avanza el modelo se mantiene en el mismo estado o avanza hacia uno de mayor índice. Esto significa que se puede asociar al tiempo con los estados de una manera directa ya que este tipo de HMM tiene la propiedad de modelar señales cuyas características cambian con el tiempo, por ejemplo, el habla. Además, podemos pensar al significado físico de los estados como sonidos propios (por ejemplo, fonos) de la palabra modelada.

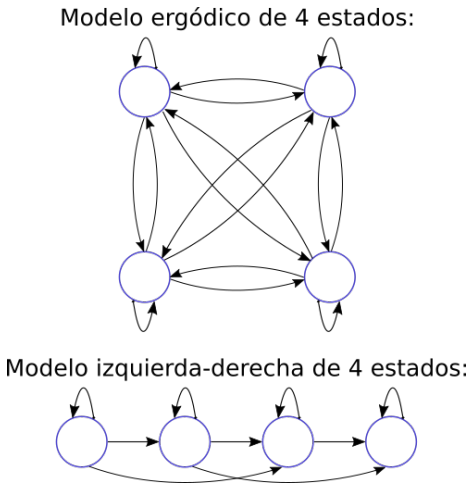

Figura 6: 2 tipos de modelos HMM.

Por otro lado, existen por lo menos dos criterios para elegir la cantidad de estados a utilizar en los modelos. Uno es hacer que la cantidad de estados coincida aproximadamente con la cantidad de fonemas de la palabra y el otro es hacer que coincida aproximadamente con el promedio de la cantidad de observaciones correspondientes a la palabra. Además, en el primero de los casos se suele utilizar la misma cantidad de estados en todos los modelos del sistema, con lo cual los mismos funcionan mejor cuando representan palabras con la misma cantidad de fonemas que de estados del modelo.

Todos los modelos izquierda-derecha tienen las siguientes propiedades:

$$a_{ij} = 0 \text{ para } j < i \tag{11}$$

$$\pi_i = 0 \text{ para } i \neq 1 \tag{12}$$

$$\pi_i = 1 \text{ para } i = 1 \tag{13}$$

$$a_{NN} = 1 \tag{14}$$

$$a_{Ni} = 0 \text{ para } i < N \tag{15}$$

Con frecuencia se imponen condiciones adicionales para asegurarse de que no ocurran grandes cambios en los índices de estados accedidos:

$$a_{ij} = 0 \text{ para } j > i + \Delta \tag{16}$$

Por otro lado, si bien existe la posibilidad de considerar a las observaciones como símbolos discretos y de esa forma utilizar densidades de probabilidad discretas para cada estado del modelo, el problema con ese abordaje es que en reconocimiento del habla las observaciones corresponden a señales continuas. Esto significa que es preferible utilizar HMMs con densidades de observación continuas.

La función de densidad de probabilidad más comúnmente utilizada es una mezcla (sumatoria) de densidades, como la siguiente [**8**], [**9**], [**10**]:

$$b_i(O) = \sum_{m=1}^{M} c_{im} . f(O, \mu_{im}, \Sigma_{im}) \text{ para } 1 \leq i \leq N \tag{17}$$

donde O es la secuencia de vectores de observaciones a modelar, c_{im} es un coeficiente para la m-esima mezcla en el estado i y f es en general una función de densidad Gaussiana con vector de media μ_{im} y matriz de covarianza Σ_{im}. Los coeficientes c_{im} satisfacen las condiciones probabilísticas:

$$\sum_{m=1}^{M} c_{im} = 1 \text{ para } 1 \leq i \leq N \tag{18}$$

$$c_{im} \geq 0 \text{ para } 1 \leq i \leq N \text{ y } 1 \leq m \leq M \tag{19}$$

de forma que la función de densidad de probabilidad quede correctamente normalizada:

$$\int_{-\infty}^{\infty} b_i(x)\, dx = 1 \text{ para } 1 \leq i \leq N \tag{20}$$

Las diferentes mezclas pueden ser consideradas como una forma especial de subestados donde las probabilidades de transición son los coeficientes c_{im}:

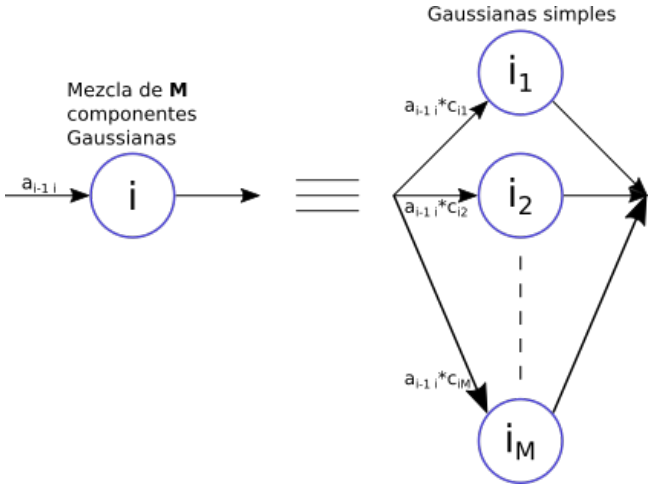

Figura 7: Representación de las mezclas de Gaussianas como subestados.

2.3. Entrenamiento: algoritmo Baum-Welch

Una vez elegido el tipo de HMM a utilizar, consideramos el problema de ajustar los parámetros del modelo $\lambda = (A, B, \pi)$ de modo de maximizar $P(O|\lambda_i)$ para cada modelo (para cada palabra) a partir del set de entrenamiento.

Para cada palabra del diccionario, dado un número finito de repeticiones de secuencias de vectores de observaciones destinadas al entrenamiento, no existe lamentablemente una forma óptima de estimar los parámetros del modelo correspondiente a dicha palabra. Sin embargo, podemos encontrar un modelo λ de manera de maximizar localmente $P(O|\lambda_i)$ utilizando un método iterativo llamado algoritmo Baum-Welch. El mismo consiste en la aplicación de los siguientes pasos:

1. Se asignan valores iniciales al modelo $\lambda = (A, B, \pi)$. Se deben tener en cuenta las propiedades de los modelos izquierda-derecha descriptas anteriormente, las cuales se mantendrán a lo largo del proceso de reestimación. Como trabajaremos con HMMs con densidades de observación continuas, podemos interpretar como que cada valor de la matriz B se compone a su vez, para cada mezcla de Gaussianas utilizada, de los valores de coeficiente de la mezcla, vector de media de la mezcla y matriz de covarianza[4] de la mezcla.

   Una pregunta fundamental al respecto, es cómo escoger los valores iniciales del modelo, de forma tal que el máximo local coincida con el máximo global de la función a optimizar. La experiencia ha mostrado que valores iniciales aleatorios funcionan bien para los valore de A y π. Sin embargo, para los valores de B se obtienen mejores resultados si primero se realiza una segmentación de las secuencias de observaciones a sus estados correspondientes [**11**]. Esta segmentación es explicada con mayor detalle más adelante.

2. Procedimiento *forward*: definimos a la variable *forward* como la probabilidad de estar en el estado S_i en el instante t y haber visto la secuencia parcial de vectores de observaciones $o_1 o_2 \ldots o_t$ dado el modelo λ:

$$\alpha_t(i) = P(o_1\ o_2 \ldots\ o_t,\ q_t = S_i|\lambda) \tag{21}$$

   Podemos hallar los valores de dicha variable de la siguiente manera (requiriendo una cantidad de cálculos en el orden de $N^2.T$):

   a)

$$\alpha_1(i) = \pi_i.b_i(O_1) \text{ para } 1 \leq i \leq N \tag{22}$$

   b)

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i).a_{ij}\right].b_j(O_{t+1}) \text{ para } 1 \leq t \leq T-1 \text{ y } 1 \leq j \leq N \tag{23}$$

3. Procedimiento *backward*: definimos a la variable *backward* como la probabilidad de ver la secuencia parcial de vectores de observaciones $o_{t+1} o_{t+2} \ldots o_T$ luego de estar en el estado S_i en el instante t dado el modelo λ:

$$\beta_t(i) = P(o_{t+1}\ o_{t+2} \ldots\ o_T|q_t = S_i, \lambda) \tag{24}$$

[4] Una de las principales ventajas de los coeficientes MFCC es que suelen no estar correlacionados, lo cual permite utilizar matrices de covarianza diagonales en los HMMs y ahorrar memoria en la implementación [**6**].

Podemos hallar los valores de dicha variable de la siguiente manera (requiriendo una cantidad de cálculos en el orden de $N^2.T$):

a)

$$\beta_T(i) = 1 \text{ para } 1 \leq i \leq N \tag{25}$$

b)

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}.b_j(O_{t+1}).\beta_{t+1}(j) \quad \text{para } t = T-1,\ T-2,\ \ldots,\ 1 \text{ y } 1 \leq i \leq N \tag{26}$$

4. Actualización: Comenzamos definiendo la variable

$$\gamma_t(i) = P(q_t = S_i|O,\lambda) = \frac{\alpha_t(i).\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j).\beta_t(j)} \tag{27}$$

donde el denominador es un factor de normalización de modo que $\sum_{i=1}^{N} \gamma_t(i) = 1$ y de esta forma dicha variable indique una probabilidad.

Además definimos la variable,

$$\xi_t(i,\ j) = P(q_t = S_i,\ q_{t+1} = S_j|O,\lambda) = \frac{\alpha_t(i).a_{ij}.\beta_{t+1}(j).b_j(O_{t+1})}{\sum_{k=1}^{N} \alpha_t(k).\beta_t(k)} \tag{28}$$

donde nuevamente el denominador es un factor de normalización de modo que dicha variable indique una probabilidad.

Esta probabilidad puede ser calculada a partir de las variables indicadas en la siguiente figura:

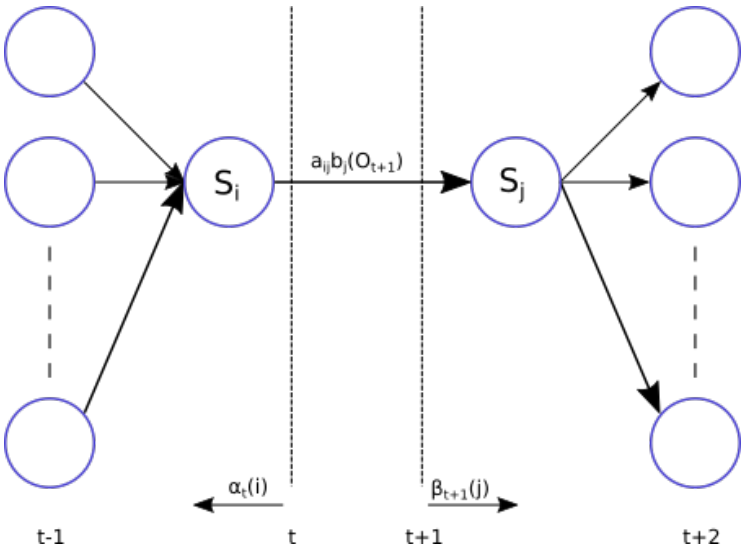

Figura 8: Variables utilizadas por el algoritmo Baum-Welch.

De esta manera, las variables recién definidas quedan relacionadas de la siguiente manera:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,\ j) \tag{29}$$

Entonces, si sumamos a lo largo de todos los instantes t obtenemos:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{ cantidad esperada de transiciones desde el estado } S_i \tag{30}$$

$$\sum_{t=1}^{T-1} \xi_t(i,\ j) = \text{ cantidad esperada de transiciones desde el estado } S_i \text{ al estado } S_j \tag{31}$$

Con estas ecuaciones, podemos proceder a obtener una actualización $\bar{\lambda}$ del modelo de la siguiente manera:

-
$$\overline{\pi_i} = \gamma_1(i) \tag{32}$$

-
$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T-1} \xi_t(i,\ j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{33}$$

- Para la actualización de la probabilidad de salida debemos considerar cada mezcla Gaussiana por separado. En lugar de $\gamma_t(i)$ utilizaremos $\gamma_t(i,k)$, es decir, la probabilidad de estar en el estado i en el instante t considerando sólo la k-ésima mezcla Gaussiana para el correspondiente vector de observaciones o_t:

$$\gamma_t(i,k) = \left[\frac{\alpha_t(i).\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j).\beta_t(j)}\right] \cdot \left[\frac{c_{ik}.f(o_t, \mu_{ik}, \Sigma_{ik})}{\sum_{m=1}^{M} c_{im}.f(o_t, \mu_{im}, \Sigma_{im})}\right] \tag{34}$$

Luego puede reestimarse cada uno de los componentes que conforman $b_i(O)$ de la siguiente manera:

$$\overline{c_{ik}} = \frac{\sum_{t=1}^{T} \gamma_t(i,k)}{\sum_{t=1}^{T} \sum_{k=1}^{M} \gamma_t(i,k)} \tag{35}$$

$$\overline{\mu_{ik}} = \frac{\sum_{t=1}^{T} \gamma_t(i,k).o_t}{\sum_{t=1}^{T} \gamma_t(i,k)} \tag{36}$$

$$\overline{\Sigma_{ik}} = \frac{\sum_{t=1}^{T} \gamma_t(i,k).(o_t - \mu_{ik}).(o_t - \mu_{ik})'}{\sum_{t=1}^{T} \gamma_t(i,k)} \tag{37}$$

Si iterativamente reemplazamos λ por $\bar{\lambda}$ y repetimos los cálculos de reestimación (pasos 2 a 4), entonces estaremos aumentando la probabilidad del modelo de observar la secuencia de vectores de observaciones O hasta alcanzar cierto límite [**12**]. El resultado final de este procedimiento de reestimación se llama estimador de máxima verosimilitud del HMM.

2.4. Reconocimiento: procedimiento forward y algoritmo Viterbi

A la hora de realizar el reconocimiento, como se mencionó previamente, para cada palabra desconocida que se pretende reconocer se debe calcular la probabilidad $P(O|\lambda)$ de cada modelo de generar la secuencia de vectores de observaciones $O = o_1, \ldots, o_T$ y luego se selecciona el modelo más probable. A su vez, suele ser útil también encontrar, para el modelo elegido, aquella secuencia de estados $Q = q_1, \ldots, q_T$ que mejor "explique" (que tenga más probabilidad de generar) la secuencia de vectores de observaciones correspondientes a la palabra desconocida.

Para resolver el primer problema se utiliza el ya explicado procedimiento *forward* incluyendo un último paso en el que se calcula la probabilidad $P(O|\lambda)$ como la suma de las variables *forward* en el último instante T, es decir:

$$\alpha_T(i) = P(o_1\ o_2 \ldots\ o_T,\ q_T = S_i|\lambda) \tag{38}$$

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{39}$$

Para resolver el segundo problema, hay distintos criterios posibles de optimización. Por ejemplo, uno podría ser el de elegir para cada instante t aquél estado q_t que sea individualmente más probable. Sin embargo, si bien de esta forma maximizamos la cantidad de estados correctos (eligiendo para cada instante t aquél estado que sea más probable), puede ser que la secuencia de estados elegida ni siquiera sea válida (ya que no se estarían teniendo en cuenta las probabilidades de transición).

La mejor solución para encontrar, de entre las secuencias de estados (caminos) válidas, aquella que sea la más probable, es aplicando el llamado algoritmo de Viterbi [**13**], [**14**], explicado a continuación.

Primero, para un determinado modelo λ, debemos definir la variable $\delta_t(i)$ como la probabilidad, en el instante t, a lo largo de la secuencia más probable de estados que termina en el estado S_i, respecto a las primeras t observaciones:

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P[q_1 q_2 \ldots q_t = i, O_1 O_2 \ldots O_t|\lambda] \tag{40}$$

Por inducción, tenemos:

$$\delta_{t+1}(j) = \left[\max_i\ \delta_t(i)\, a_{ij}\right].b_j(O_{t+1}) \tag{41}$$

Para cada instante t y estado j, determinamos el argumento que maximiza la última ecuación y lo almacenamos en $\psi_t(j)$. El procedimiento completo para encontrar la mejor secuencia de estados es entonces el siguiente:

1. Inicialización

$$\delta_1(i) = \pi_i b_i(O_1) \text{ para } 1 \leq i \leq N \tag{42}$$

$$\psi_1(i) = 0 \tag{43}$$

2. Recursión

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i)\, a_{ij}]\, b_j(O_t) \text{ para } 2 \leq t \leq T \text{ y } 1 \leq j \leq N \tag{44}$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i)\, a_{ij}] \text{ para } 2 \leq t \leq T \text{ y } 1 \leq j \leq N \tag{45}$$

3. Terminación

$$P^* = \underset{1 \leq i \leq N}{\text{máx}} \left[\delta_T (i)\right] \tag{46}$$

$$q_T^* = \underset{1 \leq i \leq N}{\text{argmax}} \left[\delta_T (i)\right] \tag{47}$$

4. Determinación en retroceso de la secuencia de estados (*backtracking*)

$$q_t^* = \psi_{t+1} \left(q_{t+1}^*\right) \text{ para } t = T-1,\ T-2,\ \ldots,\ 1 \tag{48}$$

2.5. Segmentación en estados por Viterbi/K-means segmental

Como se mencionó previamente, es importante tener unos buenos estimadores iniciales de los parámetros de las densidades $b_i(O_t)$ para lograr una buena convergencia en la reestimación de los parámetros. En la figura 9 observamos un procedimiento para obtener buenos estimadores iniciales de estos parámetros [**15**]. El procedimiento de entrenamiento es una variante del algoritmo iterativo K-means de clasificación de patrones.

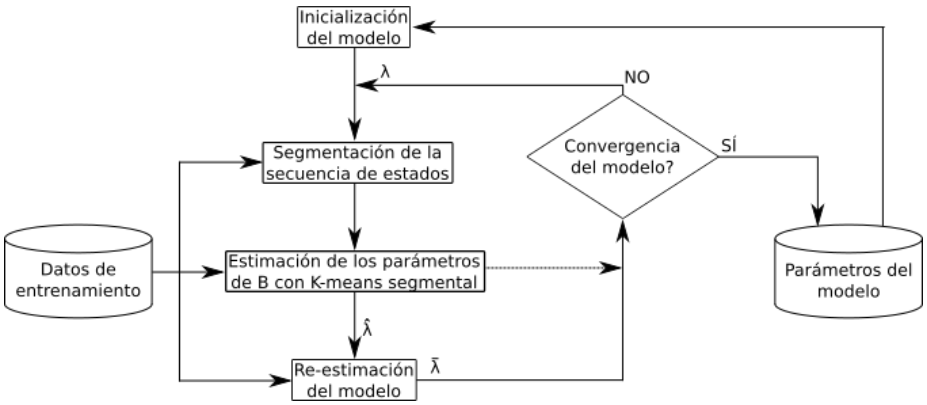

Figura 9: Procedimiento de entrenamiento con K-means segmental.

Asumimos que tenemos un set de datos de entrenamiento con observaciones (al igual que para la re-estimación de los parámetros) y un estimador inicial para todos los parámetros del modelo. Pero a diferencia del requerimiento para la re-estimación, estos estimadores iniciales pueden ser elegidos aleatoriamente.

Luego de la inicialización del modelo, el set de secuencias de observaciones destinadas para el entrenamiento es segmentada en los estados, basándose en el actual modelo λ (el modelo inicial podría ser uno creado a partir de otro set de hablantes, o bien uno creado a partir de una segmentación uniforme de cada palabra en estados). Esta segmentación es lograda encontrado la secuencia de estados optima a través del algoritmo de Viterbi. El resultado de segmentar cada una de las secuencias de observaciones es, para cada uno de los N estados, un estimador de máxima verosimilitud de aquellas observaciones que ocurren dentro de cada estado S_i, según el modelo actual.

El procedimiento de K-means segmental se utiliza para clasificar los vectores de observaciones dentro de cada estado S_i en un set de M grupos, donde cada grupo representa a una de las M mezclas de $b_i(O_t)$. Así, se obtiene un modelo actualizado $\hat{\lambda}$ al cual se le aplica el procedimiento formal de reestimación, para reestimar todos los parámetros del modelo. El modelo $\bar{\lambda}$ resultante es luego comparado con el modelo previo calculando una distancia que refleje la similitud estadística de los HMMs. Si la distancia entre los modelos excede cierto nivel, entonces el viejo modelo λ es reemplazado por el nuevo modelo (reestimado) $\bar{\lambda}$ y todo el proceso de entrenamiento es repetido. Si la distancia entre los modelos no alcanza dicho nivel, entonces se asume que hay convergencia de los modelos y se guarda el modelo con los parámetros finales.

2.6. Limitaciones de los HMM

Por la escalabilidad de su entrenamiento con grandes conjuntos de datos, sus métodos eficientes tanto para aprendizaje (entrenamiento) como para decodificación (reconocimiento), y el desempeño que alcanzan, constituyen el estado del arte desde hace más de 30 años. Sin embargo, los HMMs tienen también algunas limitaciones.

Una gran limitación es la suposición de que sucesivas observaciones son independientes entre sí y por lo tanto la probabilidad de una secuencia de observaciones puede ser escrita como el producto de probabilidades de observaciones individuales, es decir:

$$P\left(O_1\ O_2 \ldots\ O_T\right) = \prod_{i=1}^{T} P\left(O_i\right) \tag{49}$$

Otra limitación es la suposición de que las densidades de probabilidad de salida pueden ser bien representadas como una mezcla de Gaussianas. Finalmente, la propia suposición de las cadenas de Markov (la probabilidad de estar en un dado estado en el instante t sólo depende del estado en el instante $t-1$), claramente no es del todo apropiada para sonidos de habla, donde las dependencias generalmente se extienden a través de varios estados.

Además, el análisis con HMMs supone que la señal de habla es estable a lo largo de la duración de cada *frame* analizado. Esta suposición es incorrecta, debido a los procesos intrínsecos del habla, que contienen distintos grados de aleatoriedad y variabilidad temporal, incluso en segmentos de corta duración.

Sin embargo, a pesar de sus limitaciones, este tipo de modelo estadístico ha funcionado extremadamente bien para ciertos tipos de problemas de reconocimiento de habla.

3. Modelos Ocultos de Markov: implementación embebida

3.1. Algoritmo de reconocimiento a implementar en el microcontrolador

Dada una cierta secuencia de vectores de observaciones $O = o_1, \ldots, o_T$ correspondiente a una palabra que se pretende reconocer, el reconocimiento se efectúa calculando la probabilidad condicional $P\,(O|\lambda)$ para cada modelo λ. Esa probabilidad también llamada probabilidad a *posteriori* o *likelihood* indica cuán probable es que el modelo λ sea capaz de generar la secuencia de observaciones especificada. Una vez que se calculan estas probabilidades a posteriori, se selecciona como palabra reconocida a aquella cuyo modelo resulte más probable (*maximum likelihood*).

Cada modelo λ se compone de las matrices de probabilidades A (matriz de probabilidades de transiciones de estados), B (matriz de probabilidades de emisión de observaciones) y π (vector de probabilidades de estados iniciales, o *prior*).

Utilizaremos un modelo Bakis o izquierda-derecha [**7**], cuyo vector de probabilidades iniciales π tiene probabilidad 1 de comenzar en el primer estado (y 0 de comenzar en cualquier otro), y una matriz A tal que:

$$a_{ij} = 0 \text{ para } j < i \tag{50}$$

$$a_{ij} = 0 \text{ para } j > i + 1 \tag{51}$$

Es decir, al estar en un estado i, sólo pueda quedarse en ese mismo estado, o pasar al estado siguiente j.

Todos los valores de la matriz A pueden obtenerse al entrenar los modelos a partir de la base de datos etiquetada. Sin embargo, los valores de la matriz B para cada mezcla de cada estado dependen no sólo del vector de media y de la matriz de covarianza de la mezcla, que se obtienen también durante la etapa de entrenamiento, sino además de la secuencia de vectores de observaciones O específica a reconocer.

Esto significa que la etapa de reconocimiento puede separarse en dos pasos principales: primero el cálculo de la matriz B de probabilidades de salida y luego el cálculo del *likelihood* utilizando el procedimiento *forward*.

La matriz B posee dos dimensiones, cuyos tamaños corresponden a la cantidad de estados del modelo y al largo de la secuencia de vectores de observación. A su vez, en este tipo de modelos de Markov cada estado tiene asociado diferentes mezclas de Gaussianas (que las podemos pensar como "subestados"). Se ha demostrado que un estado de HMM con una función densidad de probabilidades dada por una mezcla de gaussianas es equivalente a un modelo de múltiples estados cada uno con una sola gaussiana [**10**].

Esto implica el equivalente a calcular previamente una cantidad de matrices igual a la cantidad de mezclas en los estados, y luego combinarlas para formar la matriz B. Para esto se agrega a la composición del modelo un vector de coeficientes c_{im} por cada estado, cuyos valores sumen 1, para combinar de forma ponderada los diferentes subestados que formarán cada estado del modelo.

Entonces, por cada mezcla de cada estado de cada modelo HMM y para cada instante t de la secuencia de vectores de observación se calcula el valor de la función de

densidad de probabilidad, para que en su conjunto formen la matriz B de probabilidades de salida, es decir:

$$B_{i,t}\left(O,\mu,\Sigma\right)=\sum_{m=1}^{M}c_{im}.f\left(o_t,\mu_{im},\ \Sigma_{im}\right)\ \text{para } 1\leq \text{i} \leq \text{N y } 1\leq \text{t} \leq \text{T} \qquad (52)$$

donde N es la cantidad de estados y M es la cantidad de mezclas. La función f de densidad Gaussiana que utilizaremos será la Distribución de Mezclas Gaussianas Multivariadas [**16**] [**17**], la cual se expresa de la siguiente forma:

$$f\left(o_t,\mu,\Sigma\right)=\frac{1}{\left(2\pi\right)^{\frac{d}{2}}\sqrt{\left|\Sigma\right|}}exp\left[-\frac{1}{2}\left(o_t-\mu\right)^{'}\Sigma^{-1}\left(o_t-\mu\right)\right] \qquad (53)$$

En esta expresión, las variables o_t y μ son vectores largo d (siendo d la cantidad de coeficientes utilizados) y Σ una matriz de tamaño dxd.

Una vez calculada la matriz B, se aplica el procedimiento *forward* para hallar el *likelihood* $P\left(O|\lambda\right)$ para cada modelo.

3.2. Consideraciones de implementación del algoritmo

3.2.1. Precisión numérica

El algoritmo de reconocimiento fue primero desarrollado y optimizado en Matlab y recién después implementado en el microcontrolador, lo cual permitió hacer comparaciones entre ambos y de este modo garantizar un correcto funcionamiento del sistema. Teniendo en cuenta que la FPU del microcontrolador utilizado posee precisión simple (utiliza 4 bytes para representar una variable de punto flotante), se desarrolló la implementación en Matlab de forma tal que utilice también precisión simple, ya que por default el software trabaja con precisión doble.

3.2.2. Scaling

Al analizar los cálculos implementados en el procedimiento *forward* se puede notar que cada término de la sumatoria corresponde a una productoria sobre t, de valores de las matrices A y B. Teniendo en cuenta que los números en dichas matrices representan probabilidades y que por lo tanto se encuentran dentro del rango entre 0 y 1, a medida que t aumenta, el resultado de la productoria tiende exponencialmente a 0. Este es un problema que en caso de no resolverse genera *underflow* en muchos de los cálculos del procedimiento *forward* (especialmente al estar trabajando con precisión simple en lugar de doble), lo cual tiene como consecuencia una disminución en el porcentaje de reconocimiento correcto del algoritmo. La solución a este inconveniente es incorporar un procedimiento de escalamiento [**18**].

El procedimiento es el siguiente: para cada instante t, una vez calculados los correspondientes valores de la variable *forward* $\alpha_t(i)$ para todos los N estados, multiplicarlos por un coeficiente de escalamiento calculado de la siguiente manera:

$$c_t = \frac{1}{\sum_{i=1}^{N} \alpha_t(i)} \tag{54}$$

Entonces, para un cierto instante t, la variable *forward* normalizada se calcula como:

$$\hat{\alpha}_t(i) = \alpha_t(i).c_t = \frac{\alpha_t(i)}{\sum_{i=1}^{N} \alpha_t(i)} \tag{55}$$

Si bien este procedimiento nos permite trabajar dentro del rango dinámico del microcontrolador, requiere de un cálculo diferente para el *likelihood* $P(O|\lambda)$, ya que no se pueden simplemente sumar los valores de $\hat{\alpha}_T(i)$ para los N estados, al tratarse ahora de valores escalados. Dicha probabilidad se calcula ahora como la inversa de la productoria de los T coeficientes de escalamiento:

$$\prod_{t=1}^{T} c_t . \sum_{i=1}^{N} \alpha_T(i) = \prod_{t=1}^{T} c_t . P(O|\lambda) = 1 \tag{56}$$

$$P(O|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t} \tag{57}$$

Sin embargo, dicha probabilidad podría caer nuevamente fuera del rango dinámico disponible, con lo cual, en lugar de calcularse la probabilidad, se suele trabajar con el

logaritmo de la misma, o el *log-likelihood*:

$$log\left[P\left(O|\lambda\right)\right] = -\sum_{t=1}^{T} log\left(c_t\right) \tag{58}$$

3.2.3. Implementación del algoritmo completo con probabilidades logarítmicas

Si bien el procedimiento de escalamiento es necesario para evitar caer fuera del rango dinámico disponible, lamentablemente no es suficiente ya que muchas veces las probabilidades relacionadas al cálculo de la matriz B también pueden caer fuera de dicho rango. Por lo tanto, se decidió implementar el algoritmo completo utilizando solamente probabilidades logarítmicas a lo largo del mismo.

El principal problema que se presentó al intentar utilizar el logaritmo a lo largo de todo el algoritmo, se dio al aplicarlo a una sumatoria de valores. En esos casos, se utilizó la siguiente aproximación [**19**]:

$$log\left(\sum f\left(x\right)\right) \approx log\left(max\left(f\left(x\right)\right)\right) \tag{59}$$

$$log\left(max\left(f\left(x\right)\right)\right) = max\left(log\left(f\left(x\right)\right)\right) \tag{60}$$

$$log\left(\sum f\left(x\right)\right) \approx max\left(log\left(f\left(x\right)\right)\right) \tag{61}$$

A continuación, se presentan los principales puntos del algoritmo completo implementado en forma logarítmica:

- Función de densidad de probabilidad Gaussiana:

$$\log\left(f\left(o_t, \mu, \Sigma\right)\right) = log\left(\frac{1}{\left(2\pi\right)^{\frac{d}{2}}\sqrt{|\Sigma|}} exp\left(-\frac{1}{2}\left(o_t-\mu\right)^{'}\Sigma^{-1}\left(o_t-\mu\right)\right)\right) \tag{62}$$

$$\log\left(f\left(o_t, \mu, \Sigma\right)\right) = -\frac{1}{2}\left(o_t-\mu\right)^{'}\Sigma^{-1}\left(o_t-\mu\right) - log\left(\left(2\pi\right)^{\frac{d}{2}}\sqrt{|\Sigma|}\right) \tag{63}$$

- Matriz B de probabilidades de salida:

$$log\left(B_{i,t}\left(O, \mu, \Sigma\right)\right) = log\left(\sum_{m=1}^{M} c_{im}.f\left(o_t, \mu_{im},\ \Sigma_{im}\right)\right) \tag{64}$$

$$log\left(B_{i,t}\left(O, \mu, \Sigma\right)\right) = max\left(log\left(c_{im}.f\left(o_t, \mu_{im},\ \Sigma_{im}\right)\right)\right) \tag{65}$$

$$log\left(B_{i,t}\left(O, \mu, \Sigma\right)\right) = max\left(log\left(c_{im}\right) + log\left(f\left(o_t, \mu_{im},\ \Sigma_{im}\right)\right)\right) \tag{66}$$

- Procedimiento de escalamiento:

$$log\left(c_t\right) = log\left(\frac{1}{\sum_{i=1}^{N}\alpha_t\left(i\right)}\right) = -log\left(\sum_{i=1}^{N}\alpha_t\left(i\right)\right) = -max\left(log\left(\alpha_t\left(i\right)\right)\right) \tag{67}$$

$$\begin{aligned} log\left(\hat{\alpha}_t\left(i\right)\right) &= log\left(\alpha_t\left(i\right).c_t\right) = log\left(\alpha_t\left(i\right)\right) + log\left(c_t\right) \\ &= log\left(\alpha_t\left(i\right)\right) - max\left(log\left(\alpha_t\left(i\right)\right)\right) \\ &\text{para } 1 \leq \text{i} \leq \text{N} \end{aligned} \tag{68}$$

- Procedimiento *forward*

$$log\left(\alpha_1\left(i\right)\right) = log\left(\pi_i.b_i\left(O_1\right)\right) = log\left(\pi_i\right) + log\left(b_i\left(O_1\right)\right) \text{ para } 1 \leq i \leq N \tag{69}$$

$$log\left(\alpha_{t+1}\left(j\right)\right) = log\left(\left(\sum_{i=1}^{N}\hat{\alpha}_t\left(i\right).a_{ij}\right).b_j\left(O_{t+1}\right)\right) \tag{70}$$

$$\begin{gathered} log\left(\alpha_{t+1}\left(j\right)\right) = max\left(log\left(\hat{\alpha}_t\left(i\right)\right) + log\left(a_{ij}\right)\right) + log\left(b_j\left(O_{t+1}\right)\right) \\ \text{para } 1 \leq t \leq T-1 \text{ y } 1 \leq j \leq N \end{gathered} \tag{71}$$

$$log\left[P\left(O|\lambda\right)\right] = -\sum_{t=1}^{T}\log\left(c_t\right) \tag{72}$$

3.2.4. Optimización temporal

Con el fin de obtener el mejor factor de tiempo real posible, se utilizó un archivo header con las definiciones de las variables que almacenan el resultado de $log\left(\left(2\pi\right)^{\frac{d}{2}}\sqrt{|\Sigma|}\right)$ (logaritmo del denominador de la función de densidad de probabilidad Gaussiana), con el fin de no tener que calcularlo durante el reconocimiento.

A su vez, se definen en el mismo archivo las variables que almacenan el resultado de Σ^{-1} (inversa de la matriz de covarianzas) para tampoco tener que calcularla durante el reconocimiento, mientras que Σ no se almacena ya que ya no es necesaria.

Además, con el fin principal de ahorrar memoria, para las matrices A (sólo 2 diagonales distintas de cero) y Σ^{-1} (sólo 1 diagonal distinta de cero) sólo se almacenan los valores significativos. Esto trae como ventaja adicional, que los cálculos en los que las mismas están involucradas requieren de muchas menos operaciones, lo cual disminuye el tiempo de reconocimiento.

Por último, se testearon las funciones de la librería CMSIS-DSP *arm_max_f32* (para hallar el valor máximo de un vector) y *arm_power_f32* (para elevar al cuadrado el valor de una variable), pero no fueron implementadas ya que su utilización requería de un mayor tiempo de reconocimiento. Además, se comprobó que elevar al cuadrado el valor de una variable multiplicándola explícitamente por sí misma sería más rápido que

hacerlo mediante la función *powf* (de la librería *math.h*), lo cual disminuyó notablemente el tiempo de reconocimiento, ya que dicha operación es una de las que se realiza la mayor cantidad de veces (VxQxMxT veces, siendo V la cantidad de palabras del diccionario, Q la cantidad de estados, M la cantidad de mezclas y T la cantidad de vectores de observación), al encontrarse dentro de la función de densidad de probabilidad. Cabe aclarar que al no haberse utilizado la librería CMSIS o ningún otro recurso específico de esta familia de microcontroladores, sino que todo el algoritmo fue implementado en código C, el mismo es muy fácilmente portable a cualquier tipo de procesador que se desee.

3.2.5. Optimización de memoria

Las matrices α y B del procedimiento *forward* son ambas de dos dimensiones y tienen los mismos tamaños. Esta característica, sumada a la secuencia con la que se realizan los cálculos en dicho procedimiento, permitió implementar una sola matriz (también con esos tamaños) para representarlas a ambas y así requerir la mitad de la memoria. Toda la memoria utilizada en el algoritmo corresponde a variables locales (almacenada en el stack), a excepción de esta matriz, que es alocada de forma dinámica (almacenada en el heap) ya que su tamaño depende del largo de la palabra a reconocer.

Además, el procedimiento *forward* implementado no utiliza toda la matriz de transiciones A sino sólo sus dos diagonales distintas de cero (para optimizar el uso de la memoria). Entonces, la ecuación $\sum_{i=1}^{N} \hat{\alpha}_t(i).a_{ij}$, correspondiente a la multiplicación de la matriz A completa con el vector $\hat{\alpha}$, se lleva a cabo en dos pasos, uno por cada una de las dos diagonales, y luego se suman ambos resultados.

Por último, en el cálculo de la función de densidad de probabilidad Gaussiana tampoco se utiliza toda la inversa de la matriz de covarianzas Σ^{-1} sino sólo su diagonal (es una matriz diagonal, con lo cual el resto de sus valores es cero). Entonces, la ecuación $-\frac{1}{2}(o_t - \mu)' \Sigma^{-1} (o_t - \mu)$, correspondiente al cálculo del exponente de la función, se implementa con multiplicación y acumulación de los componentes correspondientes a cada valor de dicha diagonal.

3.3. Cálculo de la memoria requerida para almacenar los modelos HMM

3.3.1. Consideraciones

- Memoria flash total disponible en el microcontrolador: 1 MB
- Se tomó como criterio que el tamaño total de los modelos no supere los 500 KB.
- Se usa la misma topología de modelo e igual cantidad de estados y mezclas para todos los modelos (un modelo por cada palabra del diccionario).
- Para la matriz de transiciones A se almacenan sólo las dos diagonales distintas de cero.
- Las matrices de covarianza Σ son diagonales y por lo tanto su tamaño es la cantidad de parámetros (coeficientes MFCC) utilizados. En lugar de almacenar los valores de la matriz, se almacenaron los valores de su inversa, para evitar realizar el cálculo de forma online.
- El tamaño de la media μ de cada mezcla también es igual a la cantidad de parámetros.
- Se almacena también el logaritmo del denominador de la función de densidad de probabilidad, para evitar realizar el cálculo de forma online.

3.3.2. Ejemplo

Variables por mezcla:

- Cantidad de parámetros: 39 (12 coeficientes MFCC, 12 delta, 12 delta-delta, 1 energía, 1 energía delta, 1 energía delta-delta). Entonces, el tamaño de μ y Σ^{-1} es también 39.

$$39\,(\mu) + 39\left(\Sigma^{-1}\right) + 1\,(\text{logDenom}) = 79 \tag{73}$$

Variables por estado:

- Cantidad de mezclas a utilizar: 2
- Cantidad de probabilidades de transición por cada estado: 2 (ej.: a_{11} y a_{12})

$$2\,(\text{transiciones}) + 79 * 2\,(\text{mezclas}) + 2\,(\text{mezclas}) = 162 \tag{74}$$

Tamaño total:

- Cantidad de estados por palabra: 16
- Cantidad de palabras a reconocer: 11
- Tamaño de variable en punto flotante: 4 bytes

$$4\,(\text{bytes}) * 11\,(\text{palabras}) * 16\,(\text{estados}) * 162 = 114048\ \text{bytes} \tag{75}$$

4. Máquinas de Vectores de Soporte: desarrollo teórico

4.1. Motivación e introducción[5]

Los HMMs son sin lugar a dudas la técnica primaria más utilizada para ASR. Durante las últimas décadas, el desarrollo de HMMs para ASR tuvo avances significativos y, como consecuencia, actualmente los HMMs son bastante adecuados para esta aplicación. Sin embargo, todavía estamos lejos de alcanzar muy altos rendimientos en sistemas ASR.

A finales de la década de 1980 y principios de la de 1990 algunos métodos alternativos fueron propuestos, algunos de los cuales estaban basados en ANNs (Redes Neuronales Artificiales). Sin embargo, a pesar de los avances conseguidos con estas alternativas, la preponderancia de los HMMs siguió siendo un hecho.

Por otro lado, durante las últimas dos décadas, apareció una nueva técnica en el área de *machine learning* (aprendizaje automático) que ha probado poder lidiar con complejos problemas de clasificación en varias áreas de aplicación: las Máquinas de Vectores de Soporte. Las SVMs son efectivos clasificadores discriminativos con varias características importantes: maximizando un margen, son capaces de lidiar con muestras de muchas dimensiones y garantizan su convergencia a un mínimo de la función costo asociada. Esto lo realizan estimando hiperplanos de decisión de manera directa, en lugar de modelar una distribución de probabilidad a partir de datos de entrenamiento, como lo hacen los HMMs.

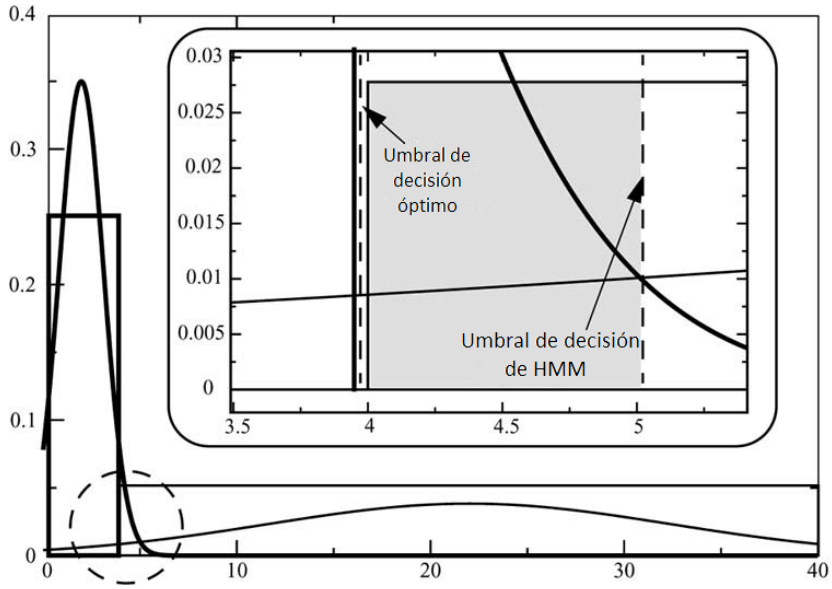

Figura 10: Ejemplo de un problema de dos clases donde la superficie de decisión encontrada para HMM no es óptima. En la vista explotada, la zona sombreada indica el error inducido al modelar datos separables con distribuciones Gaussianas (Adaptado de [**21**] y [**22**]).

[5] La subsección "Motivación e introducción" corresponde a un resumen teórico de los contenidos desarrollados en [**20**] y [**21**].

Los HMMs son modelos generativos, es decir, las decisiones a nivel acústico son tomadas basadas en la probabilidad de que el patrón evaluado haya sido generado por cada uno de los modelos que conforman el sistema de reconocimiento automático, utilizando un abordaje estadístico basado en la regla de Bayes. Sin embargo, estas decisiones son conceptualmente problemas de clasificación que podrían también ser resueltos, a veces con mejores resultados, a través de modelos discriminativos. Un simple ejemplo de esto es ilustrado en la figura 10. Las dos clases (con distribuciones rectangulares) mostradas son derivadas de distribuciones uniformes totalmente separables. En el caso de HMM, el algoritmo Baum-Welch sería utilizado para ajustar distribuciones Gaussianas (en este ejemplo, una sola mezcla de Gaussianas) a estas dos clases. Como se observa, el umbral de decisión ocurre dentro de la distribución de la clase 2, cuando debería de ocurrir sobre la frontera entre las distribuciones de ambas clases, lo cual resulta en una probabilidad de error significativa. En este ejemplo, el entrenamiento de los modelos Gaussianos nunca alcanzará la clasificación perfecta. En este sentido, los clasificadores discriminativos pueden ser clave para crear modelos más robustos y precisos.

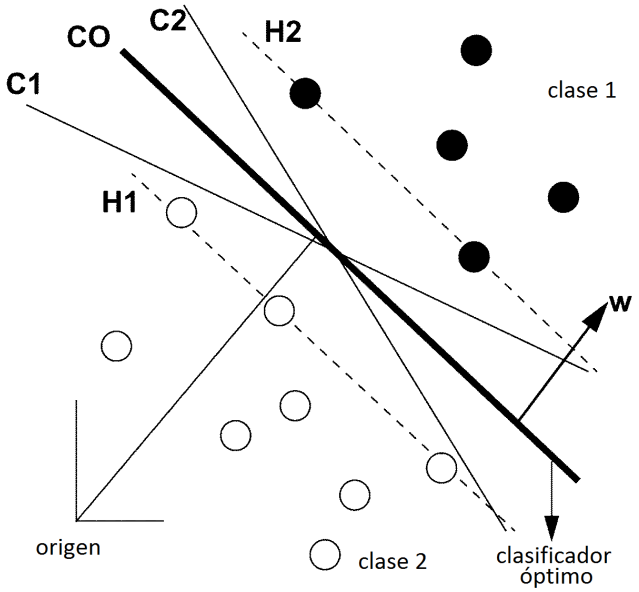

Figura 11: Diferencia entre minimización del riesgo empírico y minimización del riesgo estructural para un ejemplo simple que involucra un hiperplano clasificador. Cada hiperplano alcanza clasificación perfecta y, entonces, riesgo empírico nulo. Sin embargo, C0 es el hiperplano óptimo porque maximiza el margen (la distancia entre los hiperplanos H1 y H2). Maximizar el margen resulta indirectamente en una mejor generalización (Adaptado de [**21**]).

El funcionamiento de las SVMs se basa en maximizar un margen: la distancia entre la frontera de clasificación y las muestras. La minimización del riesgo empírico (ERM) [**23**] puede ser utilizada para encontrar un buen hiperplano, aunque no garantiza una única solución. Agregando el requisito adicional de que el hiperplano óptimo debería tener buenas propiedades de generalización[6], puede ayudar a encontrar el mejor hiperplano. La minimización del riesgo estructural (SRM) impone un ordenamiento de

[6]Utilizando una cantidad limitada de datos de entrenamiento, la mejor generalización se consigue al alcanzar la "capacidad" justa del algoritmo, es decir, la habilidad del mismo de aprender cualquier set de en-

los hiperplanos basándose en el margen. El hiperplano óptimo es aquél que maximiza el margen mientras minimiza el riesgo empírico, lo cual indirectamente garantiza una mejor generalización [**23**]. A diferencia de otras técnicas como ANNs que minimizan el riesgo empírico en el set de entrenamiento, las SVMs minimizan también el riesgo estructural, lo cual resulta en una mejor habilidad de generalizar (figura 11). En otras palabras, dado un problema de aprendizaje y un set de entrenamiento finito, las SVMs adecuadamente ponderan el potencial de aprendizaje del set y la capacidad de la máquina.

La distancia maximizada, o margen, es la responsable de las excelentes propiedades de generalización de las SVMs, ya que permite superar a la mayoría de los clasificadores no lineales en presencia de ruido, uno de los principales problemas de ASR. En un sistema libre de ruido, el margen está relacionado a la máxima distancia que una muestra correctamente clasificada debe desplazarse para ser considerada como perteneciente a la clase incorrecta. En otras palabras, indica cuánto ruido agregado a muestras "limpias" es permitido en el sistema.

trenamiento sin error. Un algoritmo con demasiada capacidad es como un botánico con memoria fotográfica, a quien cuando se le es presentado un nuevo árbol, interpreta que no es un árbol ya que tiene una cantidad de hojas distinta a cualquier cosa que haya visto antes. Un algoritmo con muy poca capacidad es como un botánico que afirma que si algo es verde, es un árbol. Ninguno de los dos generaliza bien [**24**].

4.2. Formulación matemática[7]

4.2.1. Máquinas de Vectores de Soporte lineales

4.2.1.1. Entranamiento con datos linealmente separables

Se tiene un hiperplano que separa los ejemplos positivos (pertenecientes a una clase) de los negativos (pertenecientes a la otra clase). Los vectores $\overline{x}_i$ que se encuentran sobre el hiperplano separador satisfacen $\overline{w}.\overline{x}_i + b = 0$, donde $\overline{w}$ es normal al hiperplano, $\frac{|b|}{||\overline{w}||}$ es la distancia perpendicular desde el hiperplano hasta el origen y $||\overline{w}||$ es la norma euclidea de $\overline{w}$. Si definimos a d_+ (d_-) como la distancia más corta desde el hiperplano separador al ejemplo positivo (negativo) más cercano, entonces el margen es $d_+ + d_-$. Para el caso linealmente separable, el algoritmo de vectores de soporte simplemente busca el hiperplano que separa con mayor margen.

Los vectores $\overline{x}_i$ positivos ($y_i = 1$) que se encuentran sobre el hiperplano H_1 satisfacen $\overline{w}.\overline{x}_i + b = 1$, mientras que los vectores $\overline{x}_i$ negativos ($y_i = -1$) que se encuentran sobre el hiperplano H_2 satisfacen $\overline{w}.\overline{x}_i + b = -1$. Entonces $d_+ = d_- = \frac{1}{||\overline{w}||}$ y el margen es $\frac{2}{||\overline{w}||}$. Así, para encontrar el par de hiperplanos que genera el máximo margen podemos minimizar $||\overline{w}||^2$, sujeto a la condición $y_i.\,(\overline{w}.\overline{x}_i + b) - 1 \geq 0 \;\; \forall i$. Aquellos vectores para los cuales esta ecuación da igual a cero (aquellos vectores que están ubicados sobre los hiperplanos H_1 o H_2) y cuya remoción cambiaría la solución hallada, son llamados "vectores de soporte".

Por cada vector de entrenamiento $\overline{x}_i$ existe un multiplicador de Lagrange $\alpha_i,\ i = 1, \ldots, l$, lo cual da el Lagrangiano:

$$L_P \equiv \frac{1}{2}||\overline{w}||^2 - \sum_{i=1}^{l} \alpha_i.y_i.\,(\overline{w}.\overline{x}_i + b) + \sum_{i=1}^{l} \alpha_i \tag{76}$$

Se debe minimizar L_P con respecto a $\overline{w}$ y b y simultáneamente se requiere que las derivadas de L_P respecto a todas las α_i se anulen, todo esto bajo la condición $\alpha_i \geq 0 \;\; \forall i$ (llamaremos C_1 a este set particular de condiciones). Este es un problema de programación cuadrática convexa. De manera equivalente, podemos resolver el siguiente problema "dual": maximizar L_P, sujeto a la condición de que el gradiente de L_P respecto a $\overline{w}$ y b se anule, y sujeto también a la condición $\alpha_i \geq 0 \;\; \forall i$ (llamaremos C_2 a este otro set particular de condiciones). Esta formulación dual tiene la propiedad de que el máximo de L_P, sujeto a las condiciones C_2 ocurre en los mismos valores de $\overline{w}$, b y α que el mínimo de L_P sujeto a las condiciones C_1.

Considerando que se anule el gradiente de L_P respecto a $\overline{w}$ y b, se obtiene:

$$\overline{w} = \sum_{i=1}^{l} \alpha_i.y_i.\overline{x}_i \tag{77}$$

$$\sum_{i=1}^{l} \alpha_i.y_i = 0 \tag{78}$$

Y reemplazando en la ecuación de L_P:

$$L_D = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i.\alpha_j.y_i.y_j.\overline{x_i}.\overline{x_j} \tag{79}$$

[7] La subsección "Formulación matemática" corresponde a un resumen teórico de los contenidos desarrollados en [**24**].

Se han dado a los dos Lagrangianos diferentes nombres (P por primario, D por dual), para enfatizar que las dos formulaciones son diferentes: L_P y L_D surgen de la misma función objetivo, pero con diferentes condiciones; y la solución se puede hallar minimizando L_P o maximizando L_D.

Entonces, el entrenamiento por vectores de soporte (para el caso lineal y separable), consiste en maximizar L_D respecto a los α_i, sujeto a las condiciones $\sum_{i=1}^{l} \alpha_i.y_i = 0$ y a $\alpha_i \geq 0$, y donde la solución está dada por $\overline{w} = \sum_{i=1}^{l} \alpha_i.y_i.\overline{x}_i$. En la solución, aquellos puntos para los cuales $\alpha_i > 0$ son los vectores de soporte y, como ya se mencionó, se encuentran sobre los hiperplanos H_1 o H_2.

Los vectores de soporte son elementos críticos del set de entrenamiento. Son los que se encuentran más cercanos a la frontera de clasificación, y si todos los otros vectores de entrenamiento fueran removidos (o movidos de lugar, pero sin cruzar H_1 o H_2) y el entrenamiento fuese repetido, el resultado sería exactamente el mismo hiperplano separador.

4.2.1.2. Fase de clasificación

Una vez entrenada la SVM, para utilizarla para clasificar, simplemente se determina en qué lado de la frontera de clasificación se encuentra un determinado vector incógnita $\overline{x}$ a reconocer y se le asigna la correspondiente etiqueta de clasificación. Es decir, la clase de $\overline{x}$ es $sgn\left(\overline{w}.\overline{x} + b\right)$.

4.2.1.3. Entranamiento con datos no linealmente separables

Para extender estas ideas al caso no linealmente separable, se introduce un costo a través de variables de holgura positivas $\xi_i,\ i = 1, \ldots, l$ en las condiciones, quedando:

$$y_i.\left(\overline{w}.\overline{x}_i + b\right) - 1 + \xi_i \geq 0 \tag{80}$$

$$\xi_i \geq 0 \ \ \forall i \tag{81}$$

Para que ocurra un error, la correspondiente ξ_i debe exceder la unidad, de manera que $\sum_i \xi_i$ es un límite superior en el número de esos errores de entrenamiento. Una manera natural de asignar un costo extra por errores, entonces, es cambiando la función objetivo a ser minimizada de $\frac{||\overline{w}||^2}{2}$ a $\frac{||\overline{w}||^2}{2} + C\left(\sum_i \xi_i\right)^k$, donde C es un parámetro a ser elegido por el usuario. Un C mayor, corresponde a asignar mayor penalidad a los errores.

Entonces, el entrenamiento por vectores de soporte (para el caso lineal y no separable), consiste en maximizar L_D respecto a los α_i, sujeto a las condiciones $\sum_i \alpha_i.y_i = 0$ y a $0 \leq \alpha_i \leq C$, y donde la solución está dada por $\overline{w} = \sum_{i=1}^{N_S} \alpha_i.y_i.\overline{x}_i$, donde $i = 1, \ldots, N_S$ corresponde a los índices de los vectores de soporte.

4.2.2. Máquinas de Vectores de Soporte no lineales

Para los casos en los que la función de decisión no sea función lineal de los datos, se introduce el concepto del *kernel* [**25**].

Obsérvese que la única manera en que los datos aparecen en el entrenamiento, es en la forma de productos escalares, $\overline{x}_i.\overline{x}_j$. Supongamos que primero mapeamos los datos a algún otro espacio euclideo H (incluso de infinitas dimensiones), utilizando un mapeo que llamaremos Φ:

$$\Phi : \ R^d \rightarrow H \tag{82}$$

Luego, el algoritmo de entrenamiento dependería de los datos sólo a través de productos escalares en H, es decir, en funciones de la forma $\Phi\left(\overline{x}_i\right).\Phi\left(\overline{x}_j\right)$. Entonces, si hubiera una función *kernel* K tal que $K\left(\overline{x}_i, \overline{x}_j\right) = \Phi\left(\overline{x}_i\right).\Phi\left(\overline{x}_j\right)$, sólo necesitaríamos usar K en el algoritmo de entrenamiento y ni si quiera nacesitaríamos conocer Φ. Un ejemplo es:

$$K\left(\overline{x}_i, \overline{x}_j\right) = e^{-\frac{||\overline{x}_i - \overline{x}_j||^2}{2\sigma^2}} \tag{83}$$

En este ejemplo en particular, H tiene infinitas dimensiones, con lo cual no sería fácil trabajar con Φ de forma explícita. Sin embargo, si uno reemplaza $\overline{x}_i.\overline{x}_j$ por $K\left(\overline{x}_i, \overline{x}_j\right)$ a lo largo de todo el algoritmo de entrenamiento, el mismo producirá una SVM que vive en un espacio de infinitas dimensiones. Incluso lo haría en aproximadamente el mismo tiempo que le tomaría entrenar sobre los datos no mapeados. Todas las consideraciones anteriormente explicadas se siguen cumpliendo, ya que se sigue tratando de una separación lineal, con la diferencia de que ahora es en un espacio diferente.

En la fase de clasificación se calcula el signo de:

$$f\left(\overline{x}\right) = \sum_{i=1}^{N_S} \alpha_i.y_i.\Phi\left(\overline{s}_i\right).\Phi\left(\overline{x}\right) + b = \sum_{i=1}^{N_S} \alpha_i.y_i.K\left(\overline{s}_i, \overline{x}\right) + b \tag{84}$$

donde $\overline{s}_i$ son los vectores de soporte. Entonces otra vez podemos evitar calcular $\Phi\left(\overline{x}\right)$ de manera explícita, y usar en su lugar $K\left(\overline{s}_i, \overline{x}\right) = \Phi\left(\overline{s}_i\right).\Phi\left(\overline{x}\right)$.

Si bien un clasificador SVM está definido en términos de los ejemplos de entrenamiento, como vemos, no todos los ejemplos contribuyen a la definición del clasificador. En la práctica, la proporción de vectores de soporte respecto al total de los ejemplos de entrenamiento es pequeña. Son los propios datos los que determinan cuan complejo necesita ser el clasificador, lo cual es totalmente contrario al caso de las ANNs o los HMMs, donde la complejidad del sistema suele estar predefinida antes del entrenamiento [**21**].

5. SVMs para Reconocimiento de Habla

5.1. Sistema híbrido HMM/SVM

Debido a que las SVMs no pueden modelar la variabilidad temporal del habla de manera efectiva, se mantiene el framework HMM, trabajándose entonces con modelos híbridos HMM/SVM.

El poder de una representación HMM está en su habilidad de modelar la evolución temporal de la señal a través de un proceso Markov. Esta habilidad de los HMMs de modelar estadísticamente la variabilidad acústica y temporal del habla ha sido fundamental para su éxito. La distribución de probabilidad asociada a cada estado en un HMM modela la variabilidad que ocurre en el habla a través de diferentes hablantes y contextos fonéticos [**21**].

Todo esto sigue siendo necesario al trabajar con SVMs, ya que se tratan de clasificadores estáticos (los vectores con los que se trabaja deben tener todos la misma longitud), que tienen que ser adaptados para lidiar con la duración variable en las pronunciaciones del habla.

La figura 12 muestra gráficamente el procedimiento implementado para la obtención de los vectores SVM de largo fijo, a partir de la señal de habla de largo variable. En primer lugar se realiza la extracción de parámetros MFCC a la señal acústica como ya fuera explicado previamente, obteniéndose una secuencia de vectores de observación de largo variable (a). Luego se aplica el algoritmo Viterbi para obtener la secuencia de estados más probable (b). Esto se realiza para cada uno de los modelos HMM del sistema. El siguiente paso es, para cada uno de los estados, agrupar de alguna manera los vectores de observación que fueron asociados al mismo en la secuencia de estados más probable. En este caso lo que se suele hacer (y lo que se implementó en este trabajo) es calcular el promedio de dichos vectores (c). Por último, se concatenan estos vectores promediados, quedando como resultado un vector final de largo fijo a utilizar en SVM (d). El largo de este vector es igual a la cantidad de parámetros MFCC utilizados multiplicados por la cantidad de estados de los HMMs, en nuestro caso: $39(\text{MFCC}).16(\text{estados}) = 624$.

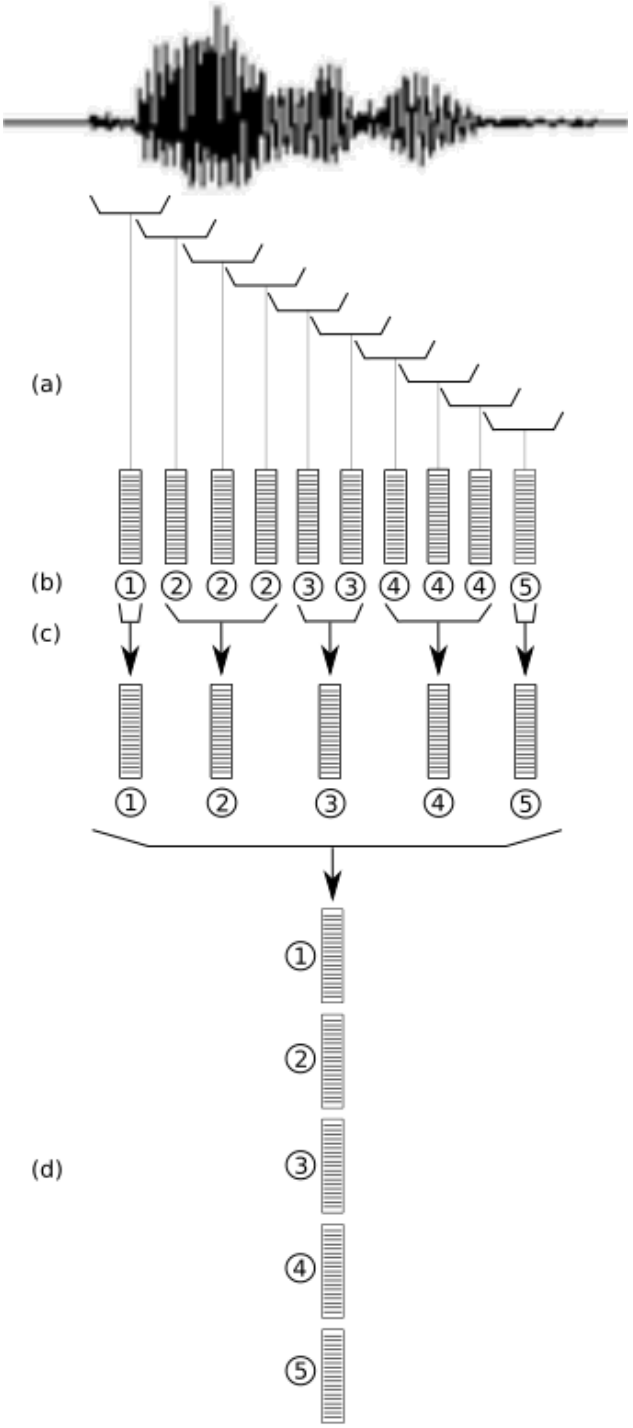

Figura 12: Procedimiento para la obtención del vector SVM a partir de la señal de habla (Ejemplo para un HMM de 5 estados).

5.2. Clasificación multi-clase

Las SVMs son en su naturaleza clasificadores del tipo binario (cada modelo discrimina entre dos clases), mientras que el ASR es un problema de clases múltiples.

Un punto fundamental en el diseño de clasificadores es si los mismos deben ser del tipo one-versus-one (uno contra uno), que aprenden a discriminar una clase de otra, o one-versus-all (uno contra todos), que aprenden a discriminar una clase de todas las demás. Los clasificadores one-versus-one son más pequeños, menos complejos y pueden ser estimados utilizando menos recursos que los clasificadores one-versus-all. Cuando el número de clases a clasificar es N, se necesita estimar $N.(N-1)/2$ clasificadores one-versus-one, en comparación con N clasificadores one-versus-all [**21**]. Si bien los clasificadores one-versus-one pueden llegar a ser un poco más precisos que los clasificadores one-versus-all, se eligió en este desarrollo trabajar con clasificadores one-versus-all, para mayor eficiencia computacional y poder alcanzar el requerimiento de reconocimiento en tiempo real.

5.3. Validación Cruzada

La validación cruzada es una técnica de validación de modelos para determinar cómo van a generalizar los resutados de un análisis estadístico a un set de testeo independiente, es decir, para estimar cuán bien el modelo va a clasificar en la práctica.

En una ronda de validación cruzada se separan los datos en dos subconjuntos, se realiza un entrenamiento en uno de estos (llamado subconjunto de entrenamiento) y una validación de dicho entrenamiento en el otro (llamado subconjunto de validación). Para reducir la variabilidad, se realizan varias rondas de validación cruzada utilizando diferentes particiones del conjunto y los resultados de la validación en las distintas rondas son promediados.

Una de las principales razones para utilizar validación cruzada en lugar de validación convencional (particionar el set en dos subconjuntos de un 70 % para entrenamiento y un 30 % para testeo) es que no hay suficientes datos como para particionarlos en subconjuntos de entrenamiento y testeo sin perder precisión en la clasificación. En estos casos, una manera efectiva de estimar correctamente la precisión del modelo es usar validación cruzada [**26**].

5.3.1. K-fold Cross-Validation

En la validación cruzada de K iteraciones (K-fold Cross-Validation) el set de entrenamiento D es separado de manera aleatoria en k subconjuntos (cada uno asociado a una iteración) mutuamente exclusivos $D_1, D_2, ..., D_k$ de aproximadamente el mismo tamaño. El modelo es entrenado y validado k veces, donde en cada vez $t \in \{1, 2, ..., k\}$ es entrenado con $D \setminus D_t$ y validado con D_t. La estimación de precisión de validación cruzada es el número total de clasificaciones correctas, dividido por la cantidad de datos del set completo [**27**].

5.3.2. Leave-one-Speaker-out Cross-Validation

Un caso particular de la validación cruzada de K iteraciones que se usa en reconocimiento de habla se llama *Leave-one-Speaker-out Cross-Validation*, y se utiliza para evaluar la robustez del reconocedor ante variaciones del hablante, determinando así sus características de generalización para el caso independiente del hablante.

Para el entrenamiento del modelo en cada iteración, se dejan de lado para el entrenamiento los datos correspondientes a uno de los hablantes y se utilizan para el procedimiento de validación. El procedimiento se repite para un hablante distinto en cada una de las k iteraciones, donde k es la cantidad de hablantes en el set de entrenamiento.

En este trabajo se utilizó *Leave-one-Speaker-out Cross-Validation* para estimar los sets de parámetros correspondientes a los modelos (en particular, los parámetros asociados a los *kernels* de los modelos SVM) con los que se obtendrían mejores tasas de reconocimiento en la generalización. Una vez encontrado el mejor set de parámetros, se volvió a entrenar con ellos el modelo por última vez, pero ahora utlizando todos los datos de entrenamiento.

6. Resultados y comparación de algoritmos

6.1. Resultados HMM

A continuación se presentan los resultados del test (para el caso HMM, sin SVM) para la elección de los modelos HMM a utilizar.

El proceso de entrenamiento y testeo de los modelos HMM se realizó utilizando una base de datos en español de 11 palabras (números 'cero' a 'diez') con las siguientes características:

- Datos para entrenamiento: 11 hablantes (7 hombres, 4 mujeres), 99 pronunciaciones c/u (3 repeticiones de cada número a 3 velocidades distintas), 1089 audios en total.
- Datos para testeo: 8 hablantes (3 mujeres, 5 hombres), 33 pronunciaciones c/u (3 repeticiones de cada número), 264 audios en total.
- Todos los audios para entrenamiento y testeo fueron grabados a 16KHz, mono y 16 bits.
- Todo el test fue realizado en una cámara anecoica para minimizar la influencia de los distintos tipos y niveles de ruido presentes en los audios grabados.

Utilizando la herramienta HTK [**6**], primero se entrenaron con los datos de entrenamiento diversos modelos con distinta cantidad de estados y de mezclas de Gaussianas. Teniendo en cuenta que la memoria flash total disponible en el microcontrolador es de 1 MB, se tomó como criterio que el tamaño total de los modelos no supere los 500 KB.

Luego se realizó el testeo utilizando el algoritmo de reconocimiento implementado en el microcontrolador. Se testearon modelos con 8, 16 y 24 estados, obteniéndose los siguientes resultados:

Cant. de estados	Cant. de mezclas	Test [%]	WER [%]	Tamaño [KB]
8	2	88.64	11.36	57
	4	94.70	5.30	113
	8	93.56	6.44	226
	16	93.56	6.44	451
16	**2**	**96.21**	**3.79**	**114**
	4	96.59	3.41	228
	8	95.83	4.17	452
24	2	95.08	4.92	171
	4	95.83	4.17	340

Cuadro 1: Rendimiento de los modelos HMM en términos del Word Error Rate (WER) y tamaño requerido para diferente cantidad de estados y mezclas por estados.

Si bien, como puede observarse, los mejores resultados se obtuvieron utilizando modelos HMM con 16 estados y 4 mezclas cada uno, se eligió como tipo de modelo implementado en el prototipo final al de 16 estados y 2 mezclas, ya que la diferencia en el rendimiento es insignificante, mientras que el requerimiento de memoria es de la mitad.

6.2. Compraración HMM vs. DTW

Para obtener una evaluación relativa del rendimiento del sistema propuesto, el mismo fue comparado con un sistema que corre bajo la misma plataforma pero basado en Dynamic Time Warping (DTW) [**28**]. DTW es una técnica de *template matching* que hace uso de programación dinámica para realizar una alineación temporal entre dos pronunciaciones, para obtener luego una comparación que indique sus similitudes [**2**], [**29**], [**30**].

Mientras que utilizar más de un prototipo o template por palabra candidata podría mejorar el grado de reconocimiento de un reconocedor basado en DTW, especialmente en el caso de sistemas independientes del hablante [**31**], esa alternativa también aumenta los requisitos de memoria y tiempo de procesamiento. En [**32**], se analizaron alternativas para el uso de sólo un prototipo por palabra usando varios ejemplos de dichas palabras, y no se encontraron mejoras significativas en comparación a la alternativa de elegir cualquier ejemplo al azar como prototipo de esa palabra. Para el caso de la comparación con el reconocedor basado en DTW, se ha elegido al azar un patrón para cada palabra a reconocer.

En la comparación, el rendimiento fue evaluado en términos del grado de reconocimiento (medido con el WER), velocidad (medida usando el factor de tiempo real) y la memoria requerida para almacenar los modelos (HMM) o templates (DTW).

El factor de tiempo real (RTF) se define como [**33**]:

$$RTF = \frac{\text{Tiempo de reconocimiento}}{\text{Largo de la pronunciación}} \tag{85}$$

A diferencia de HMM, donde el tamaño de los modelos no depende de la duración de las palabras del diccionario utilizado, sino de la cantidad de estados y mezclas elegidas, en el caso de DTW, el tamaño de los templates aumenta de forma lineal con la duración de las palabras pronunciadas. Como el tamaño puede variar incluso para diferentes pronunciaciones de la misma palabra, las duraciones de las pronunciaciones grabadas fueron promediadas y el tamaño de los templates fue calculado en función de ese valor (0.69 segundos).

Algoritmo	WER [%]	RTF	Tamaño de los modelos/templates [KB]
HMM (independiente del hablante)	3.79	0.37	114
DTW (dependiente del hablante)	2.27	0.54[8]	111[9]
DTW (independiente del hablante)	34.78		

Cuadro 2: Comparación de rendimientos en términos de WER y RTF entre los sistemas HMM y DTW.

[8] Valor medio de varias repeticiones medidas.

[9] Para templates de 0.69 segundos de largo.

El cuadro 2 muestra una comparación de resultados para HMM y DTW. El análisis HMM fue realizado sólo para el caso independiente del hablante, ya que la cantidad de ejemplos de entrenamiento disponible para cada hablante no fue suficiente para entrenar sistemas dependientes del hablante. Sin embargo, el análisis DTW fue realizado tanto para el caso independiente como el dependiente del hablante.

Mientras que el reconocedor basado en DTW mostró muy buen rendimiento en el caso dependiente del hablante, muestra una clara degradación en el caso independiente del hablante. Por otro lado, el sistema independiente del hablante basado en HMM mostró un rendimiento similar a el sistema DTW dependiente del hablante, pero sin imponer el requisito de tener muestras de habla del usuario para construir el reconocedor.

Finalmente, considerando el factor de tiempo real, el sistema HMM supera al DTW, requiriendo aproximadamente 30 % menos tiempo de reconocimiento; mientras que los requisitos de memoria fueron similares para ambas implementaciones.

6.3. Resultados HMM/SVM

Para el desarrollo del modelo híbrido HMM/SVM se probaron distintos *kernels*:

- Lineal:

$$K\left(\overline{x}_i, \overline{x}_j\right) = \overline{x}_i{}^T . \overline{x}_j \tag{86}$$

- Polinómico:

$$K\left(\overline{x}_i, \overline{x}_j\right) = \left(\overline{x}_i{}^T . \overline{x}_j + r\right)^d \tag{87}$$

- RBF (Función de base radial):

$$K\left(\overline{x}_i, \overline{x}_j\right) = e^{-\frac{||\overline{x}_i - \overline{x}_j||^2}{2\sigma^2}} \tag{88}$$

Los parámetros óptimos a encontrar por medio de validación cruzada son σ, r, d y C. Además, como se observa, el *kernel* lineal es un caso particular del *kernel* polinómico.

En primer lugar se implementó el modelo con el *kernel* RBF, observándose que la cantidad de vectores de soporte que resultaban del entrenamiento era muy alta, lo cual es un indicativo de la presencia de *overfitting*. Una manera de controlar esto es aumentando el valor de C. Sin embargo, en este caso el valor de C que permitiría conseguir un modelo con una cantidad razonable de vectores de soporte resultó muy elevado, con lo cual se concluyó que el *kernel* RBF no se adapta bien a este tipo de datos.

Luego se implementó el modelo con el *kernel* polinómico (incluyendo al lineal), con el cual sí se pudo constatar que el mismo se adapta mejor a estos datos. En el cuadro 3 se pueden observar los resultados para este caso:

Kernel	Cant. de estados	Cant. de mezclas	Test [%]	WER [%]
Polinómico	8	2	79.80	20.20
		4	75.76	24.24
		8	75.25	24.75
		16	73.74	26.26
	16	2	81.31	18.69
		4	80.81	19.19
		8	84.34	15.66
	24	2	80.30	19.70
		4	81.31	18.69

Cuadro 3: Rendimiento de los modelos HMM/SVM en términos del Word Error Rate (WER) para diferente cantidad de estados y mezclas por estados.

Como se puede observar en el cuadro de resultados, el mejor porcentaje de reconocimiento se consiguió, al igual que para el caso de los modelos HMM, con 16 estados, pero con 8 mezclas en lugar de 2. Se observa también que los resultados para el modelo híbrido HMM/SVM están muy por debajo de los mismos para el caso HMM.

La implementación del algoritmo de reconocimiento HMM/SVM incluye los cálculos correspondientes al algoritmo de Viterbi -prácticamente los mismos que para el procedimiento *forward* del reconocimiento HMM- con el agregado de aquellos relacionados con la obtención del vector SVM representativo de la pronunciación a reconocer y de aquellos correspondientes al *kernel*. Esto significa que no se justifica el desarrollo

de la implementación embebida del modelo híbrido HMM/SVM, ya que su correspondiente algoritmo de reconocimiento consumiría mayores recursos del microcontrolador (en memoria de código, debido a la implementación de los algoritmos adicionales correspondientes a SVM, y de datos, debido al almacenamiento de los vectores de soporte y sobre todo a que el mejor modelo HMM/SVM requiere de 4 veces más mezclas de Gaussianas que el mejor modelo HMM), tendría un peor RTF y el WER resultante sería mayor.

7. Conclusiones

Este trabajo describe el desarrollo teórico y la implementación de un sistema de reconocimiento de habla aislada, en tiempo real e independiente del hablante, usando Modelos Ocultos de Markov y Máquinas de Vectores de Soporte. La plataforma de implementación es un microcontrolador de 32-bit ARM Cortex-M4F.

Se introdujo la teoría, los requerimientos y los detalles de la implementación embebida. La evaluación fue realizada utilizando un *corpus* multihablante compuesto por 11 palabras del español argentino, y su rendimiento en términos de precisión, velocidad y memoria requerida fue comparada con un reconocedor basado en Dynamic Time Warping, implementado previamente en la misma arquitectura.

Los resultados obtenidos con la implementación propuesta para el caso de HMM muestra altos niveles de reconocimiento, requerimientos de memoria aceptables para vocabularios de pequeño y mediano tamaño para el caso de reconocimiento de habla aislada, y una velocidad de reconocimineto compatible con requerimientos de tiempo real.

La comparación con un sistema basado en DTW mostró mediciones de factor de tiempo real y requisitos de memoria similares para ambos algoritmos. En el caso de reconocimiento indepentiente del hablante, el rendimiento en términos del WER fue mucho mejor para HMM que para DTW. Por otro lado, para reconocimiento dependiente del hablante, DTW tiene niveles de reconocimiento similares a HMM, pero con la ventaja de no requerir una extensa base de datos para entrenar sus modelos.

Para el caso del modelo híbrido HMM/SVM, se observó que el *kernel* RBF no se adapta bien a este tipo de datos, produciéndose *overfitting* (sobreajuste) sobre los datos de entrenamiento. Por otro lado, el *kernel* polinómico se adapta correctamente a este tipo de datos durante el entrenamiento, pudiéndose encontrar sus parámetros óptimos por medio de validación cruzada.

Respecto al grado de reconocimiento del modelo híbrido HMM/SVM, teniendo su algoritmo mayores requerimientos de memoria y tiempo de procesamiento, el mismo estuvo bastante por debajo que el del sistema HMM, razón por la cual no se justifica su desarrollo e implementación embebida. Sin embargo, según la bibliografía, sí tendría mayor sentido en casos de reconocimiento de habla continua segmentada en fonos, en especial con peores condiciones de relación señal a ruido, donde el sistema HMM puede disminuir notablemente su rendimiento.

8. Trabajo a futuro

El análisis de los resultados correspondientes al grado de reconocimiento de los modelos, tanto para el caso de HMM como para el de HMM/SVM, no brindan a primera vista información acerca de cuál o cuáles podrían ser las razones que generarían mejores resultados con uno u otro tipo de clasificador. Sin embargo, una de las posibles razones de la reducción del rendimiento en el modelo híbrido HMM/SVM, respecto al caso HMM, puede haberse encontrado en el método de construcción de los vectores de largo fijo a partir de la secuencia de vectores de observación de largo variable. El hecho de promediar vectores de características elimina información potencialmente útil para el reconocimiento, que por esa razón no termina siendo utilizada para discriminar entre una clase y otra por parte del clasificador SVM.

A continuación de este trabajo, se pretende explorar otras alternativas que no dejen de lado dicha información, como pueden ser el caso de las Máquinas de Vectores de Soporte Estructuradas [**34**] y/o las Redes Neuronales Recurrentes [**35**], ambos algoritmos pertenecientes al estado del arte de reconocimiento de habla y con reportes de niveles de reconocimiento muy prometedores.

9. Bibliografía

Referencias

[1] M. Schuster, ”Speech Recognition for Mobile Devices at Google,” in PRICAI 2010: Trends in Artificial Intelligence. Springer Berlin Heidelberg, 2010, pp. 8-1.

[2] Juang, B. H., & Rabiner, L. R. (2005). Automatic speech recognition–a brief history of the technology development. Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara, 1.

[3] Yaser Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin. Learning From Data, e-Chapter 8: ”Support Vector Machines,” Amlbook.com, March, 2012.

[4] Martin, James H., and Daniel Jurafsky. ”Speech and language processing.” International Edition (2000).

[5] Rabiner, Lawrence R. ”A tutorial on hidden Markov models and selected applications in speech recognition”. Proceedings of the IEEE 77.2 (1989): 257-286.

[6] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X Liu, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, P. Woodland, P. (2006). The HTK book. Cambridge University Engineering Department.

[7] Bakis, Raimo. ”Continuous speech recognition via centisecond acoustic states.” The Journal of the Acoustical Society of America 59.S1 (1976): S97-S97.

[8] Liporace, Louis. ”Maximum likelihood estimation for multivariate observations of Markov sources.” Information Theory, IEEE Transactions on28.5 (1982): 729-734.

[9] Juang, B-H. ”Maximum-Likelihood Estimation for Mixture Multivariate Stochastic Observations of Markov Chains.” AT&T technical journal 64.6 (1985): 1235-1249.

[10] Juang, Bing-Hwang, Stephen E. Levinson, and M. Mohan Sondhi. ”Maximum likelihood estimation for multivariate mixture observations of markov chains (corresp.).” Information Theory, IEEE Transactions on 32.2 (1986): 307-309.

[11] Rabiner, Lawrence R., et al. ”Some properties of continuous hidden Markov model representations.” AT&T technical journal 64.6 (1985): 1251-1270.

[12] Baum, Leonard E., and G. R. Sell. ”Growth functions for transformations on manifolds.” Pac.¡. Math 27.2 (1968): 211-227.

[13] Viterbi, Andrew J. ”Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.” Information Theory, IEEE Transactions on 13.2 (1967): 260-269.

[14] Forney Jr, G. David. ”The viterbi algorithm.” Proceedings of the IEEE 61.3 (1973): 268-278.

[15] Rabiner, Lawrence R., et al. ”Recognition of isolated digits using hidden Markov models with continuous mixture densities.” AT&T technical journal64.6 (1985): 1211-1234.

[16] Gauvain, J. L., & Lee, C. H. (1994). ”Maximum a posteriori estimation for multivariate Gaussian mixture observations of Markov chains.” IEEE transactions on speech and audio processing, 2(2), 291-298.

[17] Xuedong Huang , Alex Acero , Raj Reddy , Hsiao-Wuen Hon, Spoken Language Processing: A Guide to Theory, Algorithm, and System Development, Prentice Hall PTR, Upper Saddle River, NJ, 2001

[18] Levinson, Stephen E., Lawrence R. Rabiner, and Man Mohan Sondhi. ”An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition.” The Bell System Technical Journal 62.4 (1983): 1035-1074.

[19] Astrov, S. (2007). Optimization of algorithms for large vocabulary isolated word recognition in embedded devices (Doctoral dissertation, Technische Universität München).

[20] Solera-Ureña, Rubén, et al. ”Svms for automatic speech recognition: a survey.” Progress in nonlinear speech processing. Springer Berlin Heidelberg, 2007. 190-216.

[21] Ganapathiraju, Aravind, Jonathan E. Hamaker, and Joseph Picone. ”Applications of support vector machines to speech recognition.” IEEE Transactions on Signal Processing 52.8 (2004): 2348-2355.

[22] E. McDermott, ”Discriminative training for speech recognition,” Ph.D. dissertation, Waseda Univ., Tokyo, Japan, 1977.

[23] V. N. Vapnik, Statistical Learning Theory. New York: Wiley, 1998.

[24] Burges, Christopher JC. ”A tutorial on support vector machines for pattern recognition.” Data mining and knowledge discovery 2.2 (1998): 121-167.

[25] Schölkopf, B., & Smola, A. J. (2002). Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

[26] Seni, G., & Elder, J. F. (2010). Ensemble methods in data mining: improving accuracy through combining predictions. Synthesis Lectures on Data Mining and Knowledge Discovery, 2(1), 1-126.

[27] Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Ijcai (Vol. 14, No. 2, pp. 1137-1145).

[28] A. G. Alvarez, D. A. Evin and S. Verrastro, ”Implementation of a Speech Recognition System in a DSC,” in press.

[29] M. Müller, ”Dynamic time warping,” in Information retrieval for music and motion, 2007, p. 318.

[30] S. Salvador and P. Chan, ”FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space,” Intell. Data Anal., vol. 11, no. 5, 2007.

[31] J. Wilpon and L. Rabiner, ”A modified K-means clustering algorithm for use in isolated work recognition,” IEEE Transactions on Acoustics, Speech, and Signal Processing 33.3 (1985): 587-594.

[32] M.S. Barakat, C.H. Ritz and D.A. Stirling, ”An improved template-based approach to keyword spotting applied to the spoken content of user generated video blogs,” 2012 IEEE International Conference on Multimedia and Expo. IEEE, 2012.

[33] Yu, D. and Li D. Automatic Speech Recognition. Springer, 2012.

[34] Zhang, S. X. (2014). Structured support vector machines for speech recognition.

[35] Zhang, S. X., Zhao, R., Liu, C., Li, J., & Gong, Y. (2016, March). Recurrent support vector machines for speech recognition. In Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on (pp. 5885-5889). IEEE.

10. Apéndices

10.1. Marco Lógico

A continuación se define el marco lógico correspondiente a este trabajo, de manera de establecer objetivos claros (fin, propósito, resultados y actividades) y la jerarquía y relación causa-efecto entre ellos.

- Fin:
  - Descripción:
    - Contribuir a la formación de recursos humanos y al desarrollo de la enseñanza de esta disciplina en el país.
    - Llevar a una implementación práctica los conocimientos disponibles en el Laboratorio de Investigaciones Sensoriales (LIS) tal que puedan servir para agregar valor en los ámbitos científico y técnico, educativo, de la salud e industrial, así como brindar retroalimentación al equipo de investigación.
  - Indicadores:
    - Tesis realizadas en el LIS (grado y posgrado).
    - Implementaciones realizadas en ámbitos académico e industrial de los resultados de estudios del LIS.
  - Medios de Verificación:
    - Comisiones evaluadoras del Conicet.
  - Supuestos/riesgos:
    - Interés de parte de la industria en realizar desarrollos basados en dichos estudios.
    - Continuidad de los recursos humanos y económicos necesarios.
- Propósito:
  - Descripción:
    - Guiar el proceso de análisis, diseño y desarrollo de un sistema de reconocimiento automático del habla embebido (RAHE).
    - Obtener un framework de referencia para RAHE que pueda ser utilizado por otros investigadores para el estudio de nuevos métodos y algoritmos de reconocimiento con aplicación práctica directa.
  - Indicadores:
    - Referencias de investigadores externos a publicaciones propias.
    - Comparación del grado de avance los los proyectos de investigación del grupo según la planificación.
  - Medios de Verificación:
    - Conclusiones de investigadores externos respecto a publicaciones propias.
    - Evaluación de los grados de avance y eventual replanificación.
  - Supuestos/riesgos:

        - Desarrollo de las ideas de proyectos basadas en información de publicaciones de terceros.

- Resultados:
    - Descripción:
        - Comparación de software embebido para reconocimiento en tiempo real de habla aislada basado en HMM/SVM con otros algoritmos de reconocimiento.
    - Indicadores:
        - Medición de rendimientos finales del software implementado.
    - Medios de Verificación:
        - Comparación de dichos rendimientos con otras tesis/papers.
        - Análisis de ventajas y desventajas de esta implementación a partir de la comparación.
    - Supuestos/riesgos:
        - Rechazo del paper a publicar.
- Actividades:
    - Descripción:
        - Teoría.
        - Entrenamiento (HTK, offline).
        - Reconocimiento (microcontrolador, tiempo real).
        - Comparación.
        - Cierre.
    - Indicadores:
        - Tests propios de cada hito.
        - Prueba de diferentes modelos HMM.
    - Medios de Verificación:
        - Comparación del grado de reconocimiento con los indicados en publicaciones investigadas.
        - Comparación de tests software microcontrolador con tests software PC.
    - Supuestos/riesgos:
        - Dificultades mayores durante el desarrollo de la tesis.
        - Falta de placa o problemas con la misma.

10.2. Implementación de un sistema de reconocimiento de dígitos aislados con HTK (Hidden Markov Model Toolkit)

Antes de comenzar agregar al *.bashrc* (en el home) la linea:
PATH=$PATH:/usr/local/speechapp/htk/bin

Esto es para que encuentre los programas HCopy, HInit, etc.

También se deben colocar en el directorio en el que se desee trabajar, los directorios y archivos de configuración y scripts, agregando también los audios para entrenamiento y testeo.

El formato que utilizaremos en esta guía para los nombres de los modelos será el siguiente: hmm$v$.m. Donde v es la versión del modelo, y m es la cantidad de mezclas.

10.2.1. Preparación de los datos

Luego de crear los directorios y copiar los archivos necesarios, corremos el script *go.mfclist*, el cual crea, a partir de los archivos que encuentra en *numbersDB/train* y *numbersDB/test*, las listas de archivos *testmfc* y *trainmfc* (ubicadas en datos) a procesar por Hcopy.

```
cd datos
../scripts/go.mfclist
```

10.2.2. Parametrización con HCopy

Con el programa HCopy realizamos la extracción de parámetros acústicos de los archivos de audio de entrenamiento y de testeo. Al programa se le pasan dos parámetros:

- El archivo *config.hcopy* contiene principalmente información sobre el tipo de archivo de entrada (ej: waveform, nist, 50 µseg de período de muestreo), el tipo de archivo de salida (ej: MFCC con derivadas, 10 mseg de solapamiento, 25 mseg de duración de ventana).
- Una de las listas creadas anteriormente (*trainmfc* o *testmfc*) donde se indican las ubicaciones de los archivos .wav a parametrizar y de los archivos .mfc a crear

```
HCopy -T 1 -C ../config/config.hcopy -S trainmfc
HCopy -T 1 -C ../config/config.hcopy -S testmfc
```

Se recomienda visualizar algunos archivos .mfc usando: HList -h x.mfc

10.2.3. Inicialización con HCompV

Partiendo de un HMM prototipo (*etc/proto*) el programa HCompV genera un nuevo HMM calculando una media y una varianza global a partir de toda la base de datos de entrenamiento (de todas las palabras: cero, uno, dos, etc). Este programa recibe como parámetros la ubicación del archivo *config.common* (que indica que el tipo de archivo de salida es MFCC con derivadas), el archivo *trainlist.mfc* (que contiene la lista de los archivos .mfc de entrenamiento, con su ubicación) y las ubicaciones del archivo *proto* y donde debe guardar el nuevo prototipo (*hmm0*). También se le indica un piso de inicialización de la varianza de 0,01 para que la misma no de 0.

```
cd ../modelos
mkdir hmm0
ls ../datos/mfc/train/* >trainlist.mfc
HCompV -A -T 1 -C ../config/config.common -f 0.01 -m -S trainlist.mfc -M hmm0
../etc/proto
```

Este nuevo modelo HMM prototipo global es el que vamos a usar para inicializar el modelo de cada dígito, para lo cual hacemos una copia del mismo por cada uno.

```
cd hmm0
cp proto cero
cp proto uno
...
cp proto nueve
cp proto diez
```

Luego debe abrirse cada modelo y reescribir donde dice "proto" por el dígito correspondiente ("cero", "uno", "dos", etc).

Por último, creamos los master label files *train1mlf.mlf* y *test1mlf.mlf* utilizando el script *go.genmlf*. Los master label files sirven para indicarle a Hinit cuál es la palabra (no se fija en el nombre del archivo) y dónde comienza y dónde termina la misma en el archivo de audio.

```
cd ../../etc
../scripts/go.genmlf train >trainmlf.mlf
../scripts/go.genmlf test >testmlf.mlf
```

10.2.4. Estimación con HInit y HRest

A continuación utilizamos el programa HInit para inicializar el modelo de cada palabra aplicando Viterbi, realizando un entrenemiento supervisado (porque tiene indicado a qué estado pertenece cada *frame*) con k-means segmental. Esto lo hacemos con el script *go.hinit*, al cual se le pasa como parámetros el listado de numeros (*/etc/numeros*) y los modelos fuente (*hmm0*) y destino (*hmm1*). Este script llama, para cada número, al programa HInit pasándole el archivo *config.common*, el nombre de los archivos de salida (los modelos de cada palabra), la ubicación del master label file y la indicación respecto a si tiene que suprimir (sobreescribir) los valores de la inicialización uniforme (lo uso cuando antes usé HCompV).

```
cd ../modelos
mkdir hmm1
../scripts/go.hinit ../etc/numeros hmm0 hmm1
```

Luego usamos el programa HRest para entrenar a los modelos haciendo una estimación con el algoritmo no supervisado (porque no tiene indicado a qué estado pertenece cada *frame*, sino que los agrupa) EM/Baum-Welch. Esto se realiza con el script *go.hrest*, que tiene las mismas opciones que *go.hinit* pero sin la indicación para suprimir los valores anteriores. El resultado de HInit (*hmm1*) se utiliza como entrada de HRest y el resultado de este es un nuevo modelo *hmm2*.

```
mkdir hmm2
../scripts/go.hrest ../etc/numeros hmm1 hmm2
```

10.2.5. Reconocimiento con HVite y evaluación con HResults

Primero creamos el diccionario de pronunciaciones y utilizamos el programa HParse para crear la red *diginet* de posibles combinaciones de modelos en una frase (esto cobra sentido en reconocimiento continuo).

```
cd ../etc
paste numeros numeros >dict
cd ../rec
HParse digitos diginet
```

Luego usamos el programa HVite para realizar el reconocimiento de los audios de test, al cual hay que pasarle los modelos HMM disenados, la lista de los archivos de coeficientes .mfc de los audios de test (*testlist.mfc*), el archivo de configuración *config.common* (con información sobre los coeficientes utilizados), el archivo *diginet* y el archivo donde se desea que HVite entrege los resultados (*results*).

```
ls ../datos/mfc/test/* >testlist.mfc
HVite -A -T 1 -C ../config/config.common -w diginet -d ../modelos/hmm2 -i results
-S testlist.mfc ../etc/dict ../etc/numeros
```

Finalmente, el programa HResults compara el resultado de HVite con la etiqueta verdadera (indicada en el master label file) y a partir de esta comparación crea una tabla indicando los porcentajes de palabras correctamente reconocidas.

```
HResults -n -A -T 1 -I ../etc/testmlf.mlf ../etc/numeros results
```

10.2.6. Refinamiento de los modelos usando mezcla de gaussianas

El programa HHEd toma como entrada un set de modelos HMM y como salida entrega un nuevo set modificado. Mediante el archivo *editf2g* le indicamos que genere un nuevo set (*hmm2.2*) de características similares al de entrada, pero que utilice 2 mezclas de gaussianas. Luego corremos nuevamente el script *go.hrest* para hacer una nueva reestimación del modelo (*hmm3.2*).

```
cd ../modelos
mkdir hmm2.2
mkdir hmm3.2
HHEd -C ../config/config.hhed -M hmm2.2 -d hmm2 editf2g ../etc/numeros
../scripts/go.hrest ../etc/numeros hmm2.2 hmm3.2
```

De la misma forma que hicimos anteriormente, utilizamos los programas Hvite y Hresults para evaluar el resultado de estos nuevos modelos.

```
cd ../rec
HVite -A -T 1 -C ../config/config.common -w diginet -d ../modelos/hmm3.2 -i re-
sults2 -S testlist.mfc ../etc/dict ../etc/numeros
```

HResults -n -A -T 1 -I ../etc/testmlf.mlf ../etc/numeros results2

Repetimos nuevamente el proceso, esperando obtener mejores porcentajes de reconocimiento a medida que vayamos aumentando la complejidad del modelo. Notar que en este caso utilizamos el archivo *editf4g* ya que deseamos obtener 4 mezclas de gaussianas.

cd ../modelos
mkdir hmm3.4
mkdir hmm4.4 HHEd -C ../config/config.hhed -M hmm3.4 -d hmm3.2 editf4g ../etc/numeros
../scripts/go.hrest ../etc/numeros hmm3.4 hmm4.4
cd ../rec
HVite -A -T 1 -C ../config/config.common -w diginet -d ../modelos/hmm4.4 -i results4 -S testlist.mfc ../etc/dict ../etc/numeros
HResults -n -A -T 1 -I ../etc/testmlf.mlf ../etc/numeros results4

Repetir el proceso incrementando la cantidad de mezclas de gaussianas hasta que el porcentaje de reconocimiento comience a disminuir (debido al *overfitting*).