Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Estadística

Verificación de locutor: Estado del arte y propuesta

Autor: Javier Cocho Cuesta

Tutor: Valentín Cardeñoso Payo

Año: 2023

Resumen

La verificación de locutor es una técnica que consiste en comprobar la identidad de una persona a través de su voz. Se basa en diferentes métodos automáticos que analizan las características del habla para tomar la decisión de si la persona que habla es quien dice ser o no.

En este Trabajo de Fin de Grado, se explorarán diferentes técnicas estadísticas para la verificación de locutor. Se analizarán los enfoques más utilizados en la literatura, como el modelo de mezclas gaussianas, se realizarán varias implementaciones, y se propondrán alternativas que permitan mejorar la precisión en la verificación.

Abstract

Speaker verification is a technique that consists of verifying the identity of a person through their voice. It is based on different automatic methods that analyze the characteristics of speech to decide whether the person speaking is who they claim to be or not.

In this end-of-degree project, different statistical techniques for speaker verification will be explored. The most widely used approaches in the literature, such as the Gaussian mixture model, will be analyzed, several implementations will be performed, and alternatives will be proposed to improve the verification accuracy.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

La verificación de locutor es una técnica que se ha vuelto cada vez más importante en la era de los asistentes virtuales y la inteligencia artificial. Esta técnica se encarga de verificar automáticamente a una persona a través de su voz, utilizando algoritmos de procesamiento de señales y aprendizaje automático.

La verificación de locutor se enfoca en la tarea de clasificación binaria, donde se busca determinar si una muestra de voz pertenece a una persona en particular o no. Es distinta a la identificación de locutor, la cual tiene como objetivo reconocer la identidad de un hablante en particular sin necesidad de saber su nombre de antemano. La identificación de locutor se enfoca en la tarea de identificación multiclase, donde se busca determinar la identidad de un hablante entre varias opciones posibles [17].

La verificación de locutor es una técnica muy importante en aplicaciones de seguridad y autenticación, ya que permite verificar la identidad de un usuario de manera automatizada y sin necesidad de contraseñas u otras medidas de seguridad. También puede ser utilizada en el análisis forense de grabaciones de audio, donde se busca identificar a la persona que habló en una grabación para propósitos de investigación o evidencia en un juicio.

Además, convirtiendo la clasificación binaria a si una muestra de voz presenta cierta enfermedad o no, puede ser utilizada en el ámbito de la salud, permitiendo la detección temprana de enfermedades como el Parkinson o problemas de audición, identificando patrones en la voz de un individuo que puedan ayudar a identificar problemas de habla en etapas tempranas, lo que puede llevar a una detección y tratamiento tempranos de enfermedades. Esto, sumado a que

puede ser una alternativa menos invasiva para el paciente, puede mejorar su calidad de vida así como su capacidad para comunicarse efectivamente con el mundo que lo rodea.

En resumen, la verificación de locutor es una técnica importante y cada vez más utilizada en diferentes aplicaciones, gracias a su capacidad de automatizar procesos de autenticación y personalización. A través de esta técnica, se busca mejorar la eficiencia y la seguridad en diferentes ámbitos, permitir la creación de sistemas de interacción humano-máquina más avanzados, y mejorar la calidad de vida de las personas.

1.1. Objetivos

Los objetivos de este trabajo de fin de grado son:

- Realizar una revisión de diversas técnicas de verificación de locutor y explicar su funcionamiento teórico.
- Realizar alguna implementación de sistemas tanto de verificación como de clasificación.
- Relacionar los problemas de verificación y clasificación y ver cómo uno de ellos puede servir para resolver el otro.

1.2. Esquema de la memoria

La memoria de este trabajo de fin de grado se organiza de la siguiente manera:

- En el Capítulo 2 se encuentra el diseño de un sistema automático de verificación de locutor. Para ello, se verá cómo dicho sistema extrae las características de una muestra de audio y cómo las procesa para a partir de ellas decidir si la muestra de audio pertenece a un locutor genuino o no.
- En el Capítulo 3 se ven diversas medidas de calidad que se utilizan para medir la fiabilidad de un sistema automático de verificación.

- En el Capítulo 4 se muestran y explican las implementaciones de varios sistemas de verificación y clasificación realizados con el lenguaje de programación Python.
- Finalmente, en el Capítulo 5, se habla sobre varias aplicaciones y usos futuros que pueden derivarse de estos sistemas.

1.3. Herramientas utilizadas

A continuación se muestra el *software* utilizado para el trabajo:

- Lucidchart: software en línea utilizado para realizar diagramas.
- Overleaf: editor de LaTeX para la realización de la memoria.
- Python: lenguaje de programación utilizado para realizar la implementación de los sistemas de verificación y clasificación.
- Visual Studio Code: entorno de desarrollo utilizado en este trabajo para programar en Python.

Capítulo 2

Diseño de un sistema automático de verificación de locutor

En los sistemas automáticos de verificación de locutor (ASV), un locutor que dice ser una persona conocida es aceptada o rechazada en función de la autenticidad de la muestra de voz que otorga. El modelo de verificación comprueba la muestra y decide si es genuina o si ha sido generada por un impostor, basándose en un valor umbral previamente establecido.

Los ASV pueden dividirse en dos grupos: sistemas dependientes de texto (TD-ASV) y sistemas independientes de texto (TI-ASV). Los TD-ASV requieren como entrada, además de una muestra de voz y la identidad del locutor, una palabra o frase solicitada por el propio sistema. Sin embargo, en los TI-ASV, el locutor puede introducir cualquier palabra o frase, y con eso el sistema podrá verificar su identidad.

El diseño de un sistema de verificación por voz está formado por dos módulos (mostrados en **2.1**): *frontend* y *backend*. El *frontend* se encarga de extraer las características (que son medidas acústicas de la voz, las cuales veremos más adelante) una vez recibida la señal de la voz. Una vez extraídas, estas son mandadas al *backend*, que compara el modelo basado en estas características con el que tiene del locutor genuino.

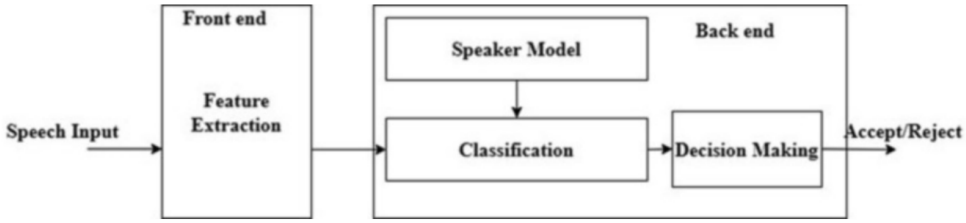


Figura 2.1: *Diagrama de un sistema de verificación de usuario [4]*

La señal de entrada es preprocesada mediante ventanas y detección de actividad de voz (VAD). La extracción de características se lleva a cabo con los *frames* de la muestra de voz obtenidos en el preprocesamiento. A continuación, estas características se modelan utilizando técnicas como el modelo de mezclas Gaussiano (GMM) o redes neuronales, que finalmente tomarán la decisión de aceptación o rechazo.

2.1. Extracción de características (*frontend*)

Dentro de un sistema automático de verificación de locutor, la función del *frontend* es la de extraer las características de la muestra de voz introducida por el locutor. Estas pueden dividirse en características basadas en el comportamiento y en características fisiológicas. Las primeras son aquellas relacionadas con el ritmo, la semántica o la pronunciación, mientas que las segundas están relacionadas con la anatomía del tracto y cuerdas vocales.

Las técnicas más utilizadas para la extracción de características se explican a continuación.

2.1.1. Coeficientes cepstrales de frecuencia mel

Los coeficientes cepstrales de frecuencia mel (Mel-frequency cepstral coefficients, MFCCs) es una técnica de extracción de características basada en la Escala Mel, que representa mejor la percepción auditiva humana que la escala de frecuencias clásica.

La percepción auditiva humana no es lineal, pero se puede aproximar utilizando funciones lineales. Dada una frecuencia f, su equivalente en la Escala Mel se calcula de la siguiente forma [2][23]:

$$mel(f) = 2595 * log_{10}(1 + f/700) \approx 1127 * ln(1 + f/700)$$

Para extraer los MFCCs, se siguen los siguientes pasos [16]:

1. La señal de audio, debido a que su contenido espectral varía con el tiempo por su naturaleza no estacionaria, se divide en segmentos temporales llamados *frames* de 10-30 ms que se solapan en un 50 %. Esto se realiza con ventanas con forma de campana; la razón de esta forma es reducir el impacto de la ventana en el espectro. Una ventana es una función matemática que se utiliza en el análisis y procesamiento de señales para evitar, en este caso, discontinuidades en los extremos de los *frames*. Habitualmente, se utiliza la ventana hamming:

   $$v(n) \approx 0{,}54 - 0{,}46 * \cos \frac{2\pi n}{N-1}, \quad 0 \leq n \leq N,$$

   siendo N el tamaño de la ventana en su forma discreta

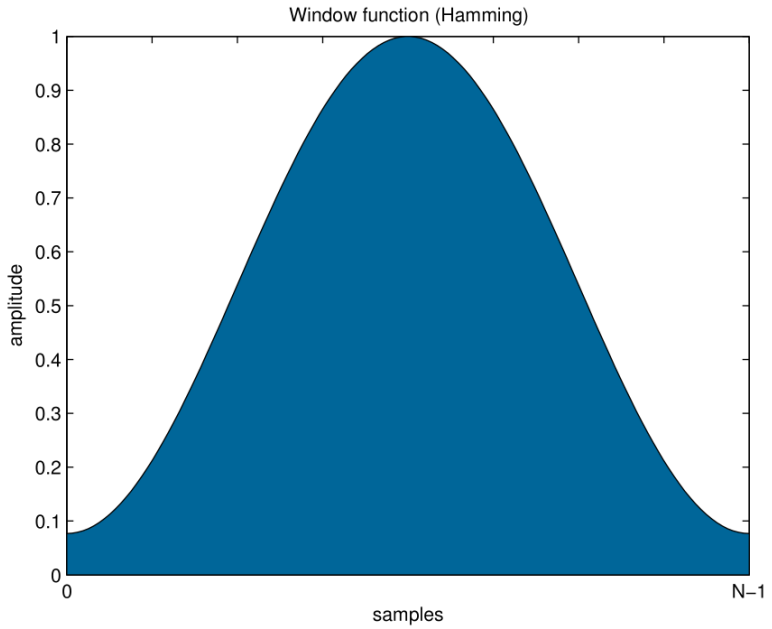


Figura 2.2: *Ventana hamming [22]*

2. El espectro de potencia de la señal a la que se le ha aplicado la ventana se calcula mediante o bien una transformada discreta de Fourier (DFT) o, si se busca eficiencia computacional, una transformada rápida de Fourier (FFT).

   Debido a que cualquier onda puede ser reconstruida sumando en cada

instante de tiempo varias ondas sinusoidales simples, se puede realizar un análisis de Fourier para calcular las frecuencias e intensidades de dichas ondas sinusoidales simples. Estas frecuencias e intensidades son las que forman el espectro de potencia de la señal.

3. El espectro de potencia de la señal es multiplicado por un banco de filtros con forma triangular (habitualmente 26 filtros). Por banco de filtros entendemos a un conjunto de filtros que separan la señal en varios componentes, en los que cada uno de ellos contiene un subconjunto de la señal original [24]. Cada uno de estos filtros se solapa en un 50 % con sus dos filtros vecinos.

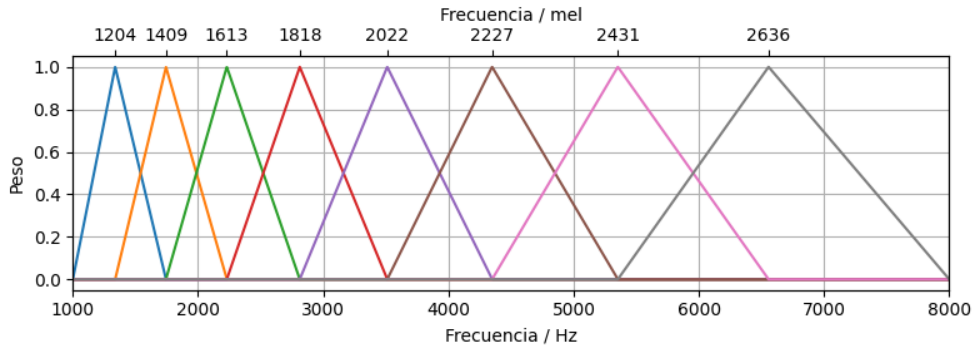


Figura 2.3: *Ejemplo de un banco de 8 filtros triangulares*

Las intensidades de las salidas del banco de filtros se escalan de forma logarítmica.

4. Se ajusta una transformada de coseno discreta (DCT) a la salida del banco de filtros. Una DCT es similar a una DFT; la principal diferencia está en que una DCT trabaja solamente con cosenos.
   En un ajuste de una DCT se seleccionan valores de los pesos de cada componente que mejor se ajustan a los datos. Si se utilizan solamente los primeros pesos de la DCT ajustada, se obtiene una versión suavizada de los datos. Los últimos pesos capturan el ruido estadístico de los datos originales. Si se utilizan todos los pesos, se pueden recuperar los valores de los datos originales.

5. Los primeros catorce pesos de la DCT ajustada forman un vector de MFCCs.

6. La ventana avanza en el tiempo, habitualmente la mitad del intervalo de la ventana (suponiendo una ventana de 20 ms, avanzaría 10 ms).

Se van repitiendo los pasos (2), (3), (4), (5) y (6) hasta que se han extraído las características de todos los *frames*.

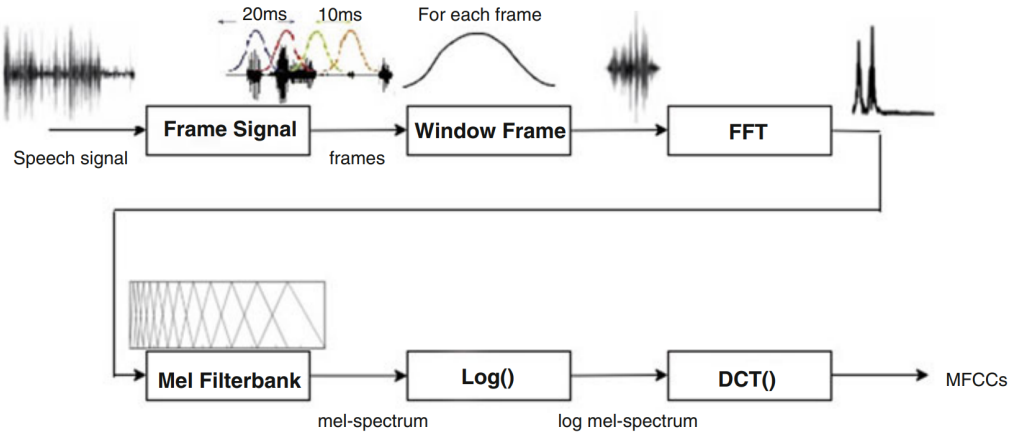


Figura 2.4: *Esquema de extracción de los MFCCs [4]*

2.1.2. Deltas y dobles deltas

Los MFCCs describen la forma espectral en un instante de la señal de audio. Debido a que la señal de audio es una señal variable en el tiempo y a que el habla es una secuencia concatenada de fonemas, surge la necesidad de recoger esa variación a lo largo del tiempo.

Los deltas aproximan las primeras derivadas de los MFCCs y recogen la tasa de cambio local de los valores de los MFCCs a lo largo del tiempo. Siendo c_t la característica c en el instante t, el delta se define como:

$$\Delta_t = c_t - c_{t-1}$$

Los dobles deltas aproximan las segundas derivadas de los MFCCs y recogen la tasa de cambio global de los deltas. El doble delta se define como:

$$\Delta\Delta_t = \Delta_t - \Delta_{t-1}$$

A cada vector de MFCCs se le concatenan los vectores de deltas y dobles deltas, triplicando los valores que contiene; si un vector de MFCCs contenía 14 valores, con los deltas y dobles deltas tendría 42 valores.

Con ello, ya se tendrían recogidas no solo las características del habla en un instante (MFCCs), sino también la variación entre instantes consecutivos.

2.1.3. Detección de la actividad de voz y diarización

En la señal de audio, debido a que solo se debería medir y tener en cuenta la voz del locutor de interés, hay que separar su voz de otros sonidos, como ruidos de la calle o la voz de otra persona, así como de los periodos de silencio. Esto se puede llevar a cabo tanto de forma automática como de forma manual, marcando el principio y el final de cada porción de audio de interés. A cada porción de audio de interés la llamaremos *utterance*.

La detección de actividad de voz (VAD) es el proceso de encontrar los *utterances* existentes en una muestra de audio. Una tarea similar es la conocida como estimación de la probabilidad de presencia de habla (SPP) en la que, en vez de decir si en una porción de audio hay o no voz, da una probabilidad de que la haya o no. De esto puede derivar una VAD, estableciendo un valor umbral de probabilidad, seleccionando aquellas porciones que lo superen.

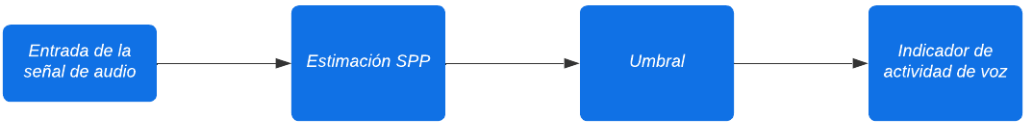


Figura 2.5: *Esquema básico de la detección automática de la actividad de voz*

La idea básica del algoritmo de VAD y SPP [3] es la siguiente:

1. Calcular una serie de características de la señal diseñadas para analizar las propiedades que diferencien el habla del resto de sonidos.

2. Juntar la información de dichas características en un clasificador, devolviendo la verosimilitud de que una porción de audio contenga voz.

3. Establecer un valor umbral para la verosimilitud del clasificador para establecer si la porción de audio contiene o no voz.

Históricamente, el clasificador más utilizado fue el árbol de decisión. Actualmente existen otros clasificadores más complejos que funcionan generalmente mejor, como modelos de mezclas gaussianas (GMM) o redes neuronales profundas (DNN). Sin embargo, estos modelos son más complicados de entrenar y aplicar, por lo que para la tarea de VAD se puede utilizar una aproximación más simple como un árbol de decisión y ver si funciona bien.

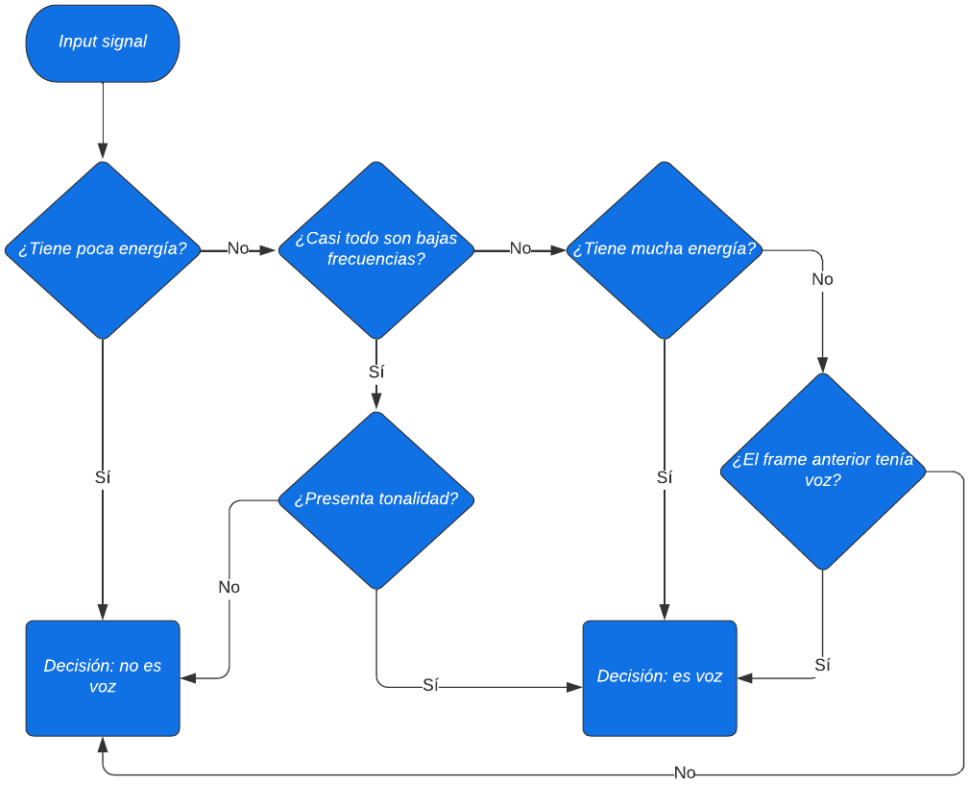


Figura 2.6: *Ejemplo de un árbol de decisión para VAD*

La VAD, aparte de facilitar la tarea de verificación eliminando porciones de audio innecesarias, también tiene otra ventaja: en el proceso de *speech enhancement* (conjunto de técnicas para mejorar la calidad de la voz respecto a otros ruidos), a la hora de reducir o eliminar el ruido de la señal, se pueden estimar las características asociadas al ruido con las porciones de audio descartadas para reducir el ruido en las *utterances*.

Por otra parte, la diarización es la técnica de diferenciar a distintos locutores de una muestra de audio o, mejor dicho, de los *utterances* de dicha muestra. La diarización responde a la pregunta *¿Quién ha hablado cuándo?* sin saber nada acerca de los locutores. Un sistema de diarización realiza las siguientes tareas:

1. Discrimina las porciones de audio que contienen voz de las que no.
2. Detecta en qué instante ha cambiado el locutor para segmentar la muestra de audio.
3. Agrupa las porciones de audio en clusters homogéneos.

En este caso, la diarización puede verse como una forma de identificación automática de locutor, con la particularidad de que no conoce a los locutores de antemano.

2.2. Compensación de desajustes en las características

Las propiedades acústicas de la voz de una persona pueden variar por diversos factores: el ruido de fondo, el tipo de micrófono con el que se recogió la muestra, la forma en la que el locutor habló (gritando, formalmente...), la distancia al micrófono... Las técnicas de compensación de desajustes buscan dos cosas: minimar las diferencias en las características cuando el locutor es el mismo, y maximizarlas cuando son diferentes.

Estas técnicas se basan en la normalización de los coeficientes de las características. Las más utilizadas se explican a continuación.

2.2.1. Sustracción de la media cepstral y normalización de la media y varianza cepstral

La sustracción de la media muestral (*cepstral-mean substraction*, CMS) [16] es una técnica de compensación de desajustes en las características que tiene como objetivo eliminar las variaciones en la respuesta espectral del habla.

Esta técnica se basa en la premisa de que una señal de habla, que cambia rápidamente con el tiempo, convoluciona con un canal que es invariante en el tiempo.

La convolución en el dominio del tiempo es equivalente a la multiplicación en el dominio de la frecuencia. Teniendo en cuenta que los MFCCs son representaciones en el dominio de la frecuencia, se pueden considerar como el resultado de multiplicar la señal del habla, que es dinámica, con la del canal, que es invariante.

Recordemos que al generar los MFCCs, estos se escalaron de forma logarítmica; por esto, se considera que los MFCCs son el resultado de sumar la señal del habla y el canal en el dominio del logaritmo de la frecuencia.

Como el canal es invariante en el tiempo, puede ser estimado obteniendo la media de los coeficientes cepstrales a lo largo del tiempo. A continuación, se resta (sustrae) la media de cada dimensión al valor existente en esa dimensión en cada *frame*. Como resultado, solo se recogen los aspectos dinámicos de las características originales de los MFCCs.

Básicamente, siendo $x(d)_{original}$ el vector de características original y $x(d)_{cms}$ el vector de características tras aplicar CMS en cada dimensión d, tenemos:

$$\mu_d = media(x(d)_{original}), \quad x(d)_{cms} = x(d)_{original} - \mu_d$$

El CMS también debe ser aplicado a los deltas y dobles deltas.

La normalización de la media y varianza cepstral (cepstral-mean-and-variance normalization, CMVN), además de lo que realiza CMS, ajusta la varianza de cada dimensión MFCC a 1.

Estas dos técnicas, además de eliminar el efecto del canal invariante, también eliminan la media de la señal del habla. Esto suele tener un comportamiento mejor a la hora de comparar una muestra de audio del locutor genuino con una del locutor cuestionado cuando hay un gran desajuste entre los canales de las grabaciones de las muestras. Sin embargo, tiene un rendimiento menor si los canales no son muy distintos.

Tras tanto la normalización CMS como la CMVN, la media de los valores de las características en cada dimensión es 0. Esto permite que los modelos estadísticos que se apliquen después de estas técnicas puedan aprovecharse de varios análisis, como el estudio de la no normalidad, el de la varianza o el de correlaciones.

2.3. Diseño del *backend*

Hemos visto cómo, en el *frontend*, se extraen las características y se modifican los vectores que las representan. Ahora, en el *backend*, se procesarán esas características y se compararán con el modelo del locutor genuino.

2.3.1. Modelo de mezclas gaussianas - Modelo de fondo universal

Para la explicación de este modelo, asumiremos que la verificación de locutor no se realiza sobre *todas las personas*, sino sobre *un grupo cerrado de personas*, llamado *población relevante*. Esto tiene especial interés principalmente en el ámbito del análisis forense, cuando se aportan grabaciones de audio en un juicio como pruebas de un caso.

Cuando todas las características se encuentran en un espacio de gran dimensión, se forman clusters. Cada uno de ellos puede representarse como una distribución gaussiana, a partir de los parámetros de la media y varianza. Los datos globales pueden representarse como una mezcla de todas las distribuciones gaussianas creadas, resultando en un modelo de mezclas gaussianas (GMM), definido por un conjunto de parámetros de media, varianza y peso.

Si asumimos una distibución multivariante, cada una de las distribuciones que forman el GMM tiene en la mixtura un vector media (μ_g), una matriz de covarianzas (Σ_g) y un peso (w_g). Si denotamos por $f(x|\mu_g, \Sigma_g)$ a cada una de la distribución g, podemos calcular la función de densidad o verosimilitud (y) evaludada en x, siendo x un vector de características, de la siguiente forma [16]:

$$y = \sum_{g=1}^{G} w_g f(x|\mu_g, \Sigma_g), \quad \sum_{g=1}^{G} w_g = 1, \quad w_g \geq 0$$

Por otro lado, el modelo de fondo universal (UBM) es un modelo general independiente del locutor que contiene a la población relevante. Obtiene GMM's dependientes del locutor, adaptando la media, varianza y pesos utilizando datos específicos del objetivo. Es construido a partir de un conjunto de datos de entrenamiento junto con un conjunto de datos del objetivo para el que estamos contruyendo el sistema. Posteriormente, se evalúa su rendimiento con un conjunto de prueba.

El modelo de mezclas gaussianas - modelo de fondo universal (Gaussian mixture model - universal background model, GMM-UBM) es un modelo estocástico muy utilizado en los TD-ASV. Este modelo se construye a partir de una grabación de un locutor conocido en concreto, y ayuda a responder a las siguientes cuestiones [16]:

- ¿Cuál sería la probabilidad de obtener el vector de características de la grabación del locutor que queremos comprobar si dicho locutor fuera el genuino?
- ¿Cuál sería la probabilidad de obtener el vector de características de la grabación del locutor que queremos comprobar si dicho locutor no fuera el genuino, sino uno de la población relevante?

Las respuesta a la primera pregunta nos da el numerador de la razón de verosimilitud, y la respuesta a la segunda pregunta el denominador.

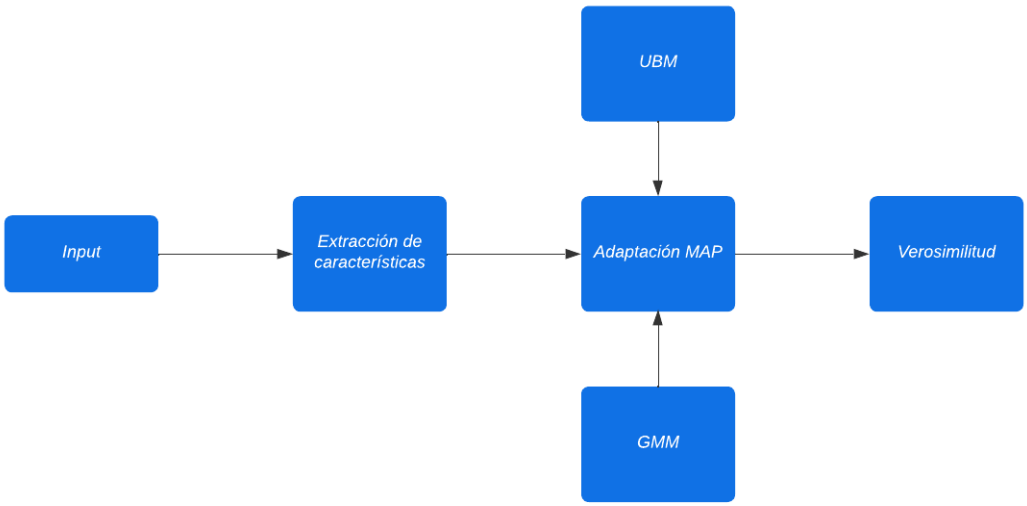


Figura 2.7: *Esquema básico de un modelo GMM-UBM*

A continuación veremos cómo se entrenan los modelos GMM-UBM.

Algoritmo esperanza-maximización

El algoritmo esperanza-maximización (algoritmo EM) se utiliza para encontrar los parámetros de máxima verosimilitud de un modelo estadístico. A menudo, se utiliza en algoritmos de agrupamiento y aprendizaje automático para

crear modelos ocultos de Markov o mixturas de gaussianas. En los problemas de verificación de locutor, se utiliza para entrenar al modelo de la población relevante, es decir, al UBM.

Asumiendo datos univariantes, para una explicación más sencilla, el algoritmo EM funciona de la siguiente manera [16]:

1. Partiendo de un número G de componentes gaussianas, se generan tantos clusters como componentes gaussianas haya, normalmente mediante un algoritmo de k-medias. Para cada una de las G componentes, se utilizan los datos del cluster en cuestión para entrenar la media inicial ($\mu_{g,0}$). Sin embargo, para el valor inicial de la varianza ($\sigma^2_{g,0}$), se utiliza la varianza de todos los datos. A todos los pesos iniciales se les suele dar el mismo valor ($w_{g,0} = 1/G$).

2. En el *expectation step*, se pregunta cómo es de probable que un nuevo dato (x_i) provenga de cada una de las componentes gaussianas; se realizan un total de G preguntas para cada x_i. Cada una de las respuestas ($\gamma_{g,i}$) se calcula como la verosimilitud relativa entre el dato x_i y la componente gaussiana g:
$$\gamma_{g,i} = \frac{w_{g,0} f(x_i|\mu_{g,0}, \sigma_{g,0})}{\sum_{j=1}^{G} w_{j,0} f(x_i|\mu_{j,0}, \Sigma_{j,0})}$$

3. En el *maximization step*, se recalculan las medias, varianzas y pesos. La nueva media y varianza de cada componente gaussiano ($\mu_{g,1}$ y $\sigma^2_{g,1}$) se calculan a partir de todos los datos multiplicando cada x_i por su peso correspondiente.
$$\mu_{g,1} = \frac{\sum_{i=1}^{N} \gamma_{g,i} x_i}{\sum_{i=1}^{N} \gamma_{g,i}}, \qquad \sigma^2_{g,1} = \frac{\sum_{i=1}^{N} \gamma_{g,i}(x_i - \mu_{g,i})^2}{\sum_{i=1}^{N} \gamma_{g,i}}$$
El nuevo peso de cada componente gaussiano ($w_{g,1}$) se calcula como la media de los $\gamma_{g,i}$ de dicho componente.
$$w_{g,1} = \frac{1}{N} \sum_{i=1}^{N} \gamma_{g,i}$$

4. Se van repitiendo los pasos (2) y (3), actualizando las medias, varianzas y los pesos para todas las componentes gaussianas. Cuando el algoritmo

converge a una solución o supera un número de iteraciones máximo establecido, para. La convergencia a una solución ocurre cuando el cambio en la bondad del ajuste del modelo a los datos de entrenamiento es menor que un valor umbral establecido previamente.

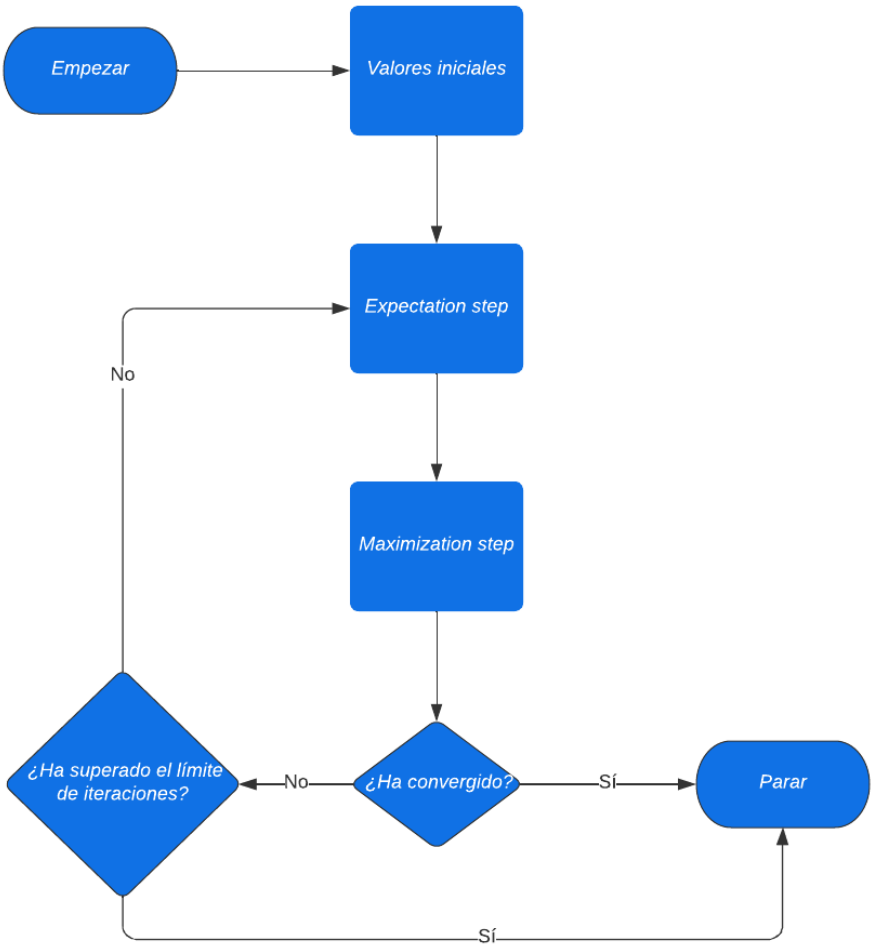


Figura 2.8: *Diagrama de flugo del algoritmo EM*

Este es el primer paso para crear un modelo GMM-UBM: entrenar un GMM de la población relevante para el problema en específico. Este se contruye a partir de los vectores de características de las muestras de las grabaciones de los locutores, que son agrupados y utilizados para crear el GMM llamado modelo de fondo universal (UBM). Este modelo se utilizará para calcular el denominador de la razón de verosimilitud.

Estimación de probabilidad máxima a posteriori

La estimación de probabilidad máxima a posteriori (MAP) es, dentro de la estadística bayesiana, un estimador muy relacionado con la estimación de máxima verosimilitud (estadística frecuentista); la principal diferencia está en que añade a la distribución y a los parámetros una probabilidad a priori que representa la información disponible anteriormente. La estimación MAP puede ser vista como una regularización de la estimación de máxima verosimilitud.

Recordemos el teorema de Bayes, utilizado mayormente para el cálculo de probabilidades condicionales:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

La cantidad que normalmente nos interesa calcular es la probabilidad a posteriori de A dado B, siendo $P(A)$ la probabilidad a priori de A.

$P(B)$, al ser una constante normalizadora, puede ser eliminada, y tendríamos una relación de proporcionalidad que quedaría de la siguiente forma:

$$P(A|B) \propto P(B|A)P(A)$$

Como estamos interesados en optimizar una cantidad, y no en estimar una probabilidad, podemos quedarnos simplemente con:

$$P(A|B) = P(B|A)P(A)$$

Ahora podemos utilizar esa fórmula para estimar la distribución y los parámetros (θ) que mejor expliquen el conjunto de datos X:

$$P(\theta|X) = P(X|\theta)P(\theta)$$

Maximizando ese valor probando con varios valores para θ se resuelve el problema [5]. Hay que tener en cuenta que el objetivo no es calcular la distribución de la probabilidad a posteriori; es calcular una estimación puntual como la moda, el valor más común. Esto se debe a que la estimación puntual ofrece una aproximación tratable, mientras que las operaciones que implican a la distribución a posteriori suelen ser poco manejables.

En el problema de verificación de locutor, la estimación MAP se utiliza para entrenar el modelo de un locutor conocido. El modelo, al igual que los anteriores, es un GMM. Para entrenarlo, en vez de partir desde cero, es adaptado

desde el UBM, debido principalmente a la alta cantidad de datos de entrenamiento que requiere un modelo GMM para funcionar correctamente, y en este caso solo se entrena con los datos de un locutor. La forma de adaptar un GMM a partir del UBM es una adaptación MAP.

La adaptación MAP funciona de la siguiente forma [16]:

1. Hay una variable α_g, llamada coeficiente de adaptación.

$$\alpha_g = \frac{N w_{g,1}}{N w_{g,1} + \tau}$$

   En esta ecuación τ es conocido como el factor de relevancia. A mayor valor de τ, menor será el del coeficiente de adaptación.

2. Se calcula una nueva media a partir de la mixtura con pesos de la media original del UBM ($\mu_{g,UBM}$) y la media que saldría como resultado si se aplicara el algoritmo EM ($\mu_{g,EM}$), junto con el coeficiente de adaptación α_g.

$$\mu_{g,1} = \alpha_g \mu_{g,EM} + (1 - \alpha_g)\mu_{g,UBM} = \alpha_g \frac{\sum_{i=1}^{N} \gamma_{g,i} x_i}{\sum_{i=1}^{N} \gamma_{g,i}} + (1 - \alpha_g)\mu_{g,0}$$

Fijándonos en la fórmula del coeficiente de adaptación podemos ver que si el peso $w_{g,1}$ de una componente gaussiana es pequeño, el coeficiente de adaptación será menor y por tanto la nueva media dependerá más de la media del UBM. Por el contrario, si $w_{g,1}$ es grande, el coeficiente de adaptación será mayor y la nueva media dependerá más de la media del EM.

Por tanto, si queremos dar más importancia a la media del UBM, podemos incrementar el valor del factor de relevancia τ; si queremos dar más importancia a la media del EM, podemos reducir dicho valor.

Todo esto puede ser interpretado como una especie de adaptación bayesiana, en la que la media del UBM equivale a la media a priori. Cuanto más estén relacionados los nuevos datos con una componente gaussiana, más cerca estará la media a posteriori de la media muestral.

Este modelo GMM es el que se utilizará para calcular el numerador de la razón de verosimilitud.

2.3.2. Razón de verosimilitud

Cálculo inicial de la razón de verosimilitud

Para calcular la razón de verosimilitud asumimos que [16]:

- Hay un modelo GMM multivariante de un locutor conocido (genuino) entrenado con las características de dicho locutor. Este modelo tiene un vector de medias μ_{g_i}, una matriz de covarianzas σ_{g_i} y pesos w_{g_i}, siendo $i = 1, 2, ..., G$.

- Tenemos los datos de un locutor cuestionado, formado por un vector de características x_{c_j} de tamaño N_c, siendo $j = 1, 2, ..., N_c$.

- Hay un segundo modelo, un UBM multivariante, entrenado con los datos de la población relevante, con vector de medias μ_{u_i}, matriz de covarianzas σ_{u_i} y pesos w_{u_i}, siendo $i = 1, 2, ..., G$.

A continuación:

1. Se calcula la verosimilitud del modelo del locutor conocido dado un vector de características del locutor cuestionado:

$$\sum_{i=1}^{G} w_{g_i} f(x_{c_j} | \mu_{g_i}, \sigma_{g_i})$$

2. Se calcula la verosimilitud del modelo UBM dado un vector de características del locutor cuestionado:

$$\sum_{i=1}^{G} w_{u_i} f(x_{c_j} | \mu_{u_i}, \sigma_{u_i})$$

3. La razón de verosimilitud $LR_{c_j,g}$ se calcula diviendo los dos términos anteriores:

$$LR_{c_j,g} = \frac{\sum_{i=1}^{G} w_{g_i} f(x_{c_j} | \mu_{g_i}, \sigma_{g_i})}{\sum_{i=1}^{G} w_{u_i} f(x_{c_j} | \mu_{u_i}, \sigma_{u_i})}$$

Realizando los tres pasos anteriores tantas veces como vectores de características haya, tenemos la razón de verosimilitud de cada uno de ellos. Sin embargo, nos interesa mostrar la evidencia de que la muestra entera de audio pertenezca o no a la del locutor genuino, no cada *frame* de forma individual.

Por ello, existe una medida que agrupa todas las verosimilitudes y calcula un valor que nos permite acercarnos más a tomar la decisión de aceptación o rechazo; la medida se llama media del logaritmo de la razón de verosimilitud.

Cálculo de la media del logaritmo de la razón de verosimilitud

La media del logaritmo de la razón de verosimilitud recoge las verosimilitudes obtenidas de cada vector de características y opera con ellas para obtener un *score* (S) con el que ayuda a tomar la decisión de si una muestra de audio pertenece al locutor genuino o no.

$$S = \frac{1}{N_g} \sum_{j=1}^{N_c} log(LR_{c_j,g})$$

La operación de sumar todos los logaritmos de las razones de verosimilitud es naive (ingenua) porque asume que no existe correlación entre las razones de verosimilitud cuando se agrupan cuando sí se espera que haya ya que, como se ha explicado anteriormente, los *frames* se solapan en un 50 % con sus adyacentes.

Además, en este caso teórico, al entrenar el UBM y los GMM de los distintos locutores, se ha estimado una gran cantidad de parámetros; recordemos que en cada vector de MFCCs hay catorce valores de la salida de una DCT más otros veintiocho para los deltas y dobles deltas. Asumiendo $G = 1024$ (valor común [16]), hay 1024 medias y 1024 varianzas, y 1023 pesos; eso da un total de 87039 parámetros $(42 * (1024 + 1024) + 1023)$. A menos que haya una cantidad muy grande de datos para entrenar los modelos, las estimaciones de los parámetros serán pobres.

Por tanto, surge un problema: el valor S puede no ser un resultado fiable a la hora de tomar la decisión de aceptación o rechazo. Por ello, se implementa lo conocido como conversión de *score* a razón de verosimilitud.

Conversión de *score* a razón de verosimilitud mediante modelos gaussianos

Recordemos que hemos convertido un conjunto de datos multivariante, todos los $LR_{c_j,g}$, en un único dato, S, que es un *score* que describe la relación entre todos los datos anteriores. Este *score* tiene que ser convertido a una única razón de verosimilitud para tomar la decisión final de aceptación o rechazo.

La conversión de *score* a razón de verosimilitud mediante modelos gaussianos es una técnica que realiza esta función. Para ello, se parte de unas muestras de entrenamiento, algunas del mismo locutor y otras de distinto locutor, y se obtienen los *scores*. Se sabe, para cada *score*, si proviene del mismo o de distinto locutor. Estos *scores* se utilizan para entrenar el modelo de mismo locutor y el de distinto locutor.

Cada uno de estos modelos es un modelo gaussiano con distinta media e igual varianza, ya que se utiliza la varianza combinada para crearlos. Con estos modelos, ya se puede calcular la razón de verosimilitud [15]:

$$LR = \frac{f(s|\mu_{ml})}{f(s|\mu_{dl})} = exp(\frac{(S-\mu_{ml})^2 - (S-\mu_{dl})^2)}{-2\sigma^2}),$$

siendo μ_{ml} y μ_{dl} las medias de los *scores* de entrenamiento de mismo locutor y distinto locutor, respectivamente.

Relación entre modelos gaussianos y regresión logística

La conversión de *score* a razón de verosimilitud mediante regresión logística es mejor que mediante modelos gaussianos. Esto es debido principalmente a que no asume que la varianza de los modelos sea igual, como sí hace la conversión mediante modelos gaussianos, lo que hace que sea más robusto [15].

Para clarificar las próximas ecuaciones, H_{ml} representará la *hipótesis de mismo locutor*, y H_{dl} la *hipótesis de distinto locutor*.

A partir de la varianza de los modelos gaussianos, se puede relacionar cada uno de ellos directamente con un modelo de regresión logística. La probabilidad de H_{ml} dado un *score* se puede calcular de a partir de la siguiente ecuación [15], una forma del teorema de Bayes:

$$p(H_{ml}|S) = \frac{f(S|H_{ml}) * p(H_{ml})}{f(S|H_{ml}) * p(H_{ml}) + f(S|H_{dl}) * p(H_{dl})}$$

siendo $p(H_{ml}|S)$ la probabilidad a posteriori de la *hipótesis de mismo locutor* modelada dado un *score* S.

Si asumimos iguales probabilidades a priori, es decir, $p(H_{ml}|S) = p(H_{dl}|S)$, podemos simplificar la ecuación anterior:

$$p(H_{ml}|S) = \frac{f(S|H_{ml})}{f(S|H_{ml}) + f(S|H_{dl})}$$

Codificando los *scores* de entrenamiento de mismo locutor como 1, y 0 los de distinto locutor, podemos modelar una curva sigmoidea mediante regresión logística. El resultado de esta codificación es un número entre 0 y 1, que indica la probabilidad de que un *score* provenga de la categoría de mismo locutor. Conviene recordar que los cálculos de una regresión logísticas se realizan en el espacio de los $log(odds)$ utilizando una técnica de máxima verosimilitud, y que este espacio de probabilidades se puede relacionar con el espacio de los $log(odds)$ de la siguiente forma:

$$log(\frac{p(H_{ml}|S)}{p(H_{dl}|S)}) = log(\frac{p(H_{ml}|S)}{1 - p(H_{ml}|S)})$$

Como hemos asumido iguales probabilidades a priori, las odds a priori son 1, y el logaritmo de la razón de verosimilitud tiene el mismo valor que el logaritmo de las odds a posteriori:

$$log(\frac{p(H_{ml}|S)}{p(H_{dl}|S)}) = log(\frac{f(S|H_{ml})}{f(S|H_{dl})}) = log(LR)$$

Conversión de *score* a razón de verosimilitud mediante regresión logística

La relación entre un *score* y el logaritmo de una razón de verosimilitud es lineal en el espacio del logaritmo de las odds; esto hace que la conversión de *scores* a logaritmos de razón de verosimilitud sea sencilla.

Siendo α y β coeficientes entrenados con regresión logística, la conversión de *score* a razón de verosimilitud es la siguiente [15][16]:

$$log(LR) = \alpha + \beta S, \quad \alpha = \frac{-\beta(\mu_{ml} + \mu_{dl})}{2}, \quad \beta = \frac{\mu_{ml} - \mu_{dl}}{\sigma^2}$$

2.4. Otras técnicas

A continuación se muestran otras técnicas utilizadas para la extracción de características y el modelado de sistemas de verificación.

2.4.1. Vector identidad

El vector identidad (i-vector), como su nombre indica, es un vector que recoge toda la información extraída de una muestra de audio de un locutor.

Un i-vector se crea de la siguiente manera [16]:

1. Se entrena un UBM a partir de muchas grabaciones de múltiples locutores. Al contrario que en los modelos GMM-UBM explicados anteriormente, estas grabaciones no representan a la población relevante: mejor cuantas más grabaciones de distintos locutores y en cuantas más condiciones distintas se recogieran.

2. Se entrena un GMM por cada grabación de los locutores de la población relevante. Al igual que en los modelos GMM-UBM, cada GMM se entrena a partir de una adaptación MAP del UBM anterior.

3. Se concatenan los vectores media de todas las componentes gaussianas en lo que se conoce como un *supervector*. Suponiendo que tenemos vectores de 14 dimensiones y 1024 componentes, cuando se concatenan todos forman dicho supervector que tiene un total de $14 * 1024 = 14336$ dimensiones.

4. Mediante análisis de componentes principales (PCA), se reducen drásticamente las dimensiones, cogiendo aquellas que recogen la mayor parte de la varianza, que son las primeras dimensiones PCA.

Los vectores resultantes tras el PCA son los i-vectores.

Recordemos que en los modelos GMM-UBM se utilizaban varios vectores para representar una grabación de un locutor genuino; en este caso, un i-vector la representa. Es por ello por lo que se requieren otras técnicas de compensación de desajustes y de modelado para obtener las razones de verosimilitud de los vectores.

Para compensación de desajustes, una de las más utilizadas es LDA (*linear discriminant analysis*), que se basa en encontrar las dimensiones que maximicen la razón entre la varianza entre categorías y la varianza dentro de una categoría.

Para el modelado se suele emplear PLDA (*probabilistic linear discriminant analysis*). Esta técnica, al igual que el GMM-UBM, también calcula una razón de verosimilitud, pero basado en dos preguntas distintas a en las que se basa el GMM-UBM.

2.4.2. Redes neuronales profundas

Para el problema de la verificación de locutor también se pueden utilizar redes neuronales profundas. Un ejemplo sería la utilización de estas junto con un sistema PLDA de i-vectores para entrenar el UBM (en vez de utilizar el algoritmo EM).

Otra posibilidad es el uso de estas para la creación de sistemas de x-vectores (sistemas *embedding*). Los x-vectores sustituyen a los i-vectores como entrada del PLDA. Estos sistemas permiten como entrada una matriz bidimensional (en vez de un vector de características) que contiene en una dimensión los MFCCs y en la otra dimensión la variable tiempo discretizada. Los sistemas de x-vectores, según experimentos realizados en [20], suelen obtener mejores resultados que los i-vectores.

Capítulo 3

Medidas de calidad

Las medidas de calidad y evaluación de un ASV se calculan en base a la tasa de falsos positivos (FP) y falsos negativos (FN) del sistema, y se miden en base al *threshold* establecido.

A continuación se muestran algunas de las medidas más utilizadas [14].

EER

Un ASV puede clasificar muestras de las siguientes formas:

- De forma correcta: aceptación verdadera (AV) y rechazo verdadero (RV).
- De forma incorrecta: aceptación falsa (AF) y rechazo falso (RF).

Una AF ocurre cuando una muestra de un locutor que no es el genuino supera el *threshold* y se le verifica como genuino, mientras que un RF ocurre cuando una muestra de un locutor genuino no supera dicho valor y no se le verifica.

Lo más deseado es tener una tasas de AV y RV altas, y de AF y de RF bajas. Estas tasas se ajustan con el valor *threshold* fijado.

La tasa de AF (TAF) se calcula de la siguiente manera:

$$TAF = \frac{suma(AF)}{suma(muestras\_impostoras)}$$

La tasa de RF se calcula como:

$$TRF = \frac{suma(RF)}{suma(muestras\_genuinas)}$$

EER (*equal error rate*) es el valor en el que coinciden ambas tasas.

Curva DET

La curva DET (*detection error tradeoff*) es una representación gráfica de EER con los distintos TAF en el eje de abscisas y TRF en el eje de ordenadas.

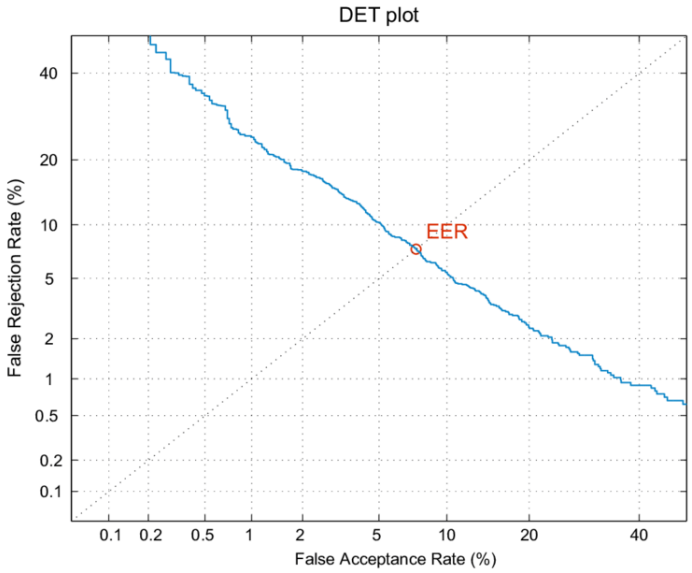


Figura 3.1: *Ejemplo de curva DET [11]*

Esta curva, utilizada en sistemas biométricos, permite identificar el punto donde coinciden ambas tasas para obtener el valor EER.

Capítulo 4

Implementación de sistemas automáticos de verificación e identificación de locutor

Se han realizado dos implementaciones con el lenguaje de programación Python para hacer pruebas prácticas tanto de un ASV como de un sistema de identificación. La primera de ellas hace uso de un modelo GMM-UBM, mientras que la segunda utiliza redes neuronales profundas.

4.1. Datos

Se ha utilizado un dataset obtenido en Kaggle [9] para realizar ambas implementaciones. Este dataset contiene el discurso de cinco políticos: Benjamin Netanyahu, Jens Stoltenberg, Julia Gillard, Margaret Thatcher y Nelson Mandela. A su vez, está dividido en cinco carpetas, cada una de ellas con el nombre de cada político.

Cada uno de estos cinco discursos está dividido en 1500 muestras de un segundo de duración, que se encuentran en la carpeta respectiva de cada político. El audio está en formato *WAV*, codificado mediante PCM con una frecuencia de muestreo de 16000 muestras por segundo.

Al ser un discurso dividido, numerosas muestras contienen solamente silencio o aplausos. Para tratar esto, se han filtrado las muestras en base a la varianza de los MFCCs: si era demasiado baja, era silencio, y si era demasiado alta, aplausos.

4.2. CRISP-DM

Para realizar las implementaciones, se ha seguido la metodología conocida como *Cross-Industry Standard Process for Data Mining* (CRISP-DM) [6], que es un enfoque estándar utilizado en la minería de datos y el análisis de datos. CRISP-DM proporciona un sistema estructurado que sirve de guía a través de las diferentes etapas del proceso de minería y análisis de datos, que va desde la comprensión del problema hasta su implementación.

El método CRISP-DM consta de las siguientes seis fases:

1. Comprensión del negocio (del trabajo): en esta etapa se busca comprender los objetivos y requisitos del trabajo; en este caso, diseñar un sistema de verificación para los datos obtenidos, así como un sistema de clasificación.

2. Comprensión de los datos: es esta etapa se realiza una exploración inicial de los datos y se valora su calidad (realizado en *Datos*).

3. Preparación de los datos: en esta etapa se limpian y transforman los datos para que sean adecuados para el análisis. En este caso, la obtención de los vectores de características de las muestras de audio.

4. Modelado: en esta etapa se aplican técnicas de modelado a los datos para obtener de ellos relaciones o patrones. En este caso, la generación del modelo GMM-UBM.

5. Evaluación: en esta etapa se evalúa la calidad de los modelos generados. En este caso, los resultados que veremos en *Resultados*.

6. Implementación: en esta etapa se pone en práctica la solución desarrollada. Esto puede ser, por ejemplo, la integración del modelos en sistemas ya existentes. No es objetivo de este trabajo realizar esta última implementación.

4.3. Implementación de un sistema automático de verificación con GMM-UBM

Para el primero de los sistemas, se ha utilizado la técnica MFCC para extraer las características de las muestras de audio y un GMM-UBM para modelarlas.

4.3.1. Librerías

Se han utilizado las siguiente librerías para la implementación de este sistema:

- *Librosa*:
  Librosa [13] es una librería de Python diseñada para el análisis de música y audio que permite, entre otras cosas, obtener los MFCCs.

- *NumPy*:
  NumPy [18] es una librería de Python que incorpora la clase de datos arrays para dar soporte a matrices grandes multidimensionales, además de disponer de funciones muy eficientes para su manipulación.

- *Scikit-learn*:
  Scikit-learn [19] es una librería de Python contruida sobre otras librerías (entre ellas *NumPy*) que contiene herramientas sencillas y eficientes para el análisis de datos predictivos.

4.3.2. Código empleado

A continuación se muestra y explica el código utilizado de forma detallada para cada una de las fases del proceso de creación del sistema.

Extracción de características

```
def extract_mfcc(audio_path, num_mfcc=14):
    audio, sr = librosa.load(audio_path)
    mfcc = librosa.feature.mfcc(y=audio, sr=sr,
        n_mfcc=num_mfcc)
    return mfcc
    features = []
for audio_path in audio_paths:
    mfcc = extract_mfcc(audio_path)
    features.append(mfcc)
```

La función *extract_ mfcc*, mediante la librería *Librosa*, extrae los MFCCs de una muestra de audio a partir de su ruta absoluta. Los MFCCs de todas las muestras de entrenamiento se guardan en un array llamado *features*.

Entrenamiento del UBM

```
ubm = GaussianMixture(n_components=128, covariance_type='diag')
ubm.fit(features)
```

Con la función *GaussianMixture* de la librería *sklearn*, se crea un ajuste de un modelo GMM para las muestras de audio de toda la población relevante. A esta función se le pasa de parámetros el número de componentes gaussianas (128) y el tipo de covarianza (*diag* ya que cada componente tiene su propia matriz de covarianzas).

Entrenamiento del GMM - Adaptación MAP

```
speaker_gmms = {}
for label in set(labels):
    index = np.where(np.array(labels) == label)[0]
    speaker_features = features[index]
    speaker_gmm =
        GaussianMixture(n_components=ubm.n_components,
        covariance_type=ubm.covariance_type, reg_covar=0.015)
    speaker_gmm.means_ = ubm.means_
    speaker_gmm.covariances_ = ubm.covariances_
    speaker_gmm.weights_ = ubm.weights_
    speaker_gmm.fit(speaker_features)
    speaker_gmms[label] = speaker_gmm
```

Con la función *GaussianMixture* de nuevo, se ajustan modelos GMM independientes para cada locutor de interés a partir de los datos del modelo UBM entrenado anteriormente (medias, covarianzas y pesos). Además, se le añade una regularización no negativa que se añade a la diagonal de la matriz de convarianzas (0.015, valor mínimo que permite 128 componentes gaussianas en los modelos independientes con estos datos). Cada uno de estos modelos se guarda en *speaker_ gmms*.

Verificación

```
def verify_speaker(audio_path, claimed_speaker):
    mfcc = extract_mfcc(audio_path)
    features = mfcc.reshape(1, -1)

    # Razon de verosimilitud para cada GMM.
    log_likelihood_ratios = []
    log_likelihood_ubm = ubm.score(features)
    for label, gmm in speaker_gmms.items():
        log_likelihood_speaker = gmm.score(features)
        # Resta porque log(a/b) = log(a) - log(b)
        log_likelihood_ratio = log_likelihood_speaker -
            log_likelihood_ubm
        log_likelihood_ratios.append((label,
            log_likelihood_ratio))

    log_likelihood_ratios.sort(key=lambda x: x[1],
        reverse=True)

    # Comprobar si tiene el mayor ratio el que dice ser.
    best_speaker, best_log_likelihood_ratio =
        log_likelihood_ratios[0]
    threshold = -50
    if best_speaker == claimed_speaker and
        best_log_likelihood_ratio > threshold:
        return True
    else:
        return False
```

La función *verify_ speaker* devuelve si un locutor dice quien dice ser o no. Para ello, a partir de una muestra de audio, extrae los MFCCs mediante la función *extract_ mfcc* explicada anteriormente y calcula los ratios de verosimilitud respecto al modelo UBM y a cada uno de los modelos GMM de cada locutor independiente.

Posteriormente, calcula el logaritmo de la razón de verosimilitud del UBM respecto a cada modelo GMM y los almacena en *log_ likelihodd_ ratios* ordenados de mayor a menor.

Finalmente, comprueba si el locutor cuestionado ha dicho que es aquel con el que corresponde el modelo que tiene mayor razón de verosimilitud y, si la verosimilitud supera el valor de *threshold* establecido (en este caso -50), el sistema acepta al locutor cuestionado. Si el locutor correspondiente al modelo que tiene mayor razón de verosimilitud no coincide con aquel que ha dicho que es el locutor, o si no supera el *threshold*, el sistema rechaza al locutor.

4.3.3. Resultados

Esta implementación es una especie de sistema de identificación y verificación a la vez: primero asigna una muestra de audio al locutor con el que tiene más verosimilitud dentro de la población relevante (clasificación) y posteriormente toma la decisión de aceptación o rechazo en base a la verosimilitud que tiene con dicho locutor.

En *Tabla 1*, *Tabla 2* y *Tabla 3* se muestran los resultados de esta implementación: *Tasa acierto* hace referencia al porcentaje de muestras bien clasificadas de cada locutor, *Tasa aceptación* al porcentaje de muestras del locutor genuino que son aceptadas por el sistema, y *Tasa error* el porcentaje de muestras del locutor que serían aceptadas como las de otro locutor del sistema.

Estos resultados corresponden a la salida del modelo de las últimas 100 muestras de cada locutor, habiendo utilizado las 1400 primeras para el entrenamiento (6139 en vez $1400 * 5 = 7000$ tras eliminar las muestras de aplausos y silencio), y estableciendo valores de *threshold* de 0 y -50 en la $log(verosimilitud)$.

A continuación se muestran las tasas de acierto del modelo sobre cada locutor:

Locutor	**Tasa acierto**
Benjamin Netanyahu	1.00
Jens Stoltenberg	0.94
Julia Gillard	0.98
Margaret Thatcher	0.97
Nelson Mandela	0.99

Tabla 4.1: *Tasa de bien clasificados en la implementación GMM-UBM*

En *Tabla 1* se puede ver que como sistema de clasificación funciona bastante bien, con un porcentaje de acierto global del 97.60 %..

En las dos siguientes tablas se muestran las tasas de aceptación y de error para los modelos con *threshold* de 0 y -50:

Locutor	**Tasa aceptación 0**	**Tasa aceptación -50**
Benjamin Netanyahu	0.63	0.92
Jens Stoltenberg	0.50	0.78
Julia Gillard	0.63	0.96
Margaret Thatcher	0.70	0.91
Nelson Mandela	0.75	0.96

Tabla 4.2: *Tasa de aceptación en la implementación GMM-UBM*

Locutor	**Tasa error 0**	**Tasa error -50**
Benjamin Netanyahu	0.00	0.00
Jens Stoltenberg	0.00	0.00
Julia Gillard	0.00	0.01
Margaret Thatcher	0.01	0.02
Nelson Mandela	0.00	0.00

Tabla 4.3: *Tasa de error en la implementación GMM-UBM*

En *Tabla 3* se puede ver que el sistema pocas veces acepta como locutor genuino a uno que no lo es (apenas el 1 % de las de *Margaret Thatcher* cuando se utiliza de valor umbral 0). Esto también hace que la tasa de aceptación no sea muy alta respecto a la de clasificación como se observa en la primera columna de *Tabla 2*; se podría incrementar esta tasa de aceptación decrementando el *threshold*, pero a cambio de que se aceptara en más ocasiones a locutores no genuinos como genuinos.

Relacionado con esto último, llama la atención que las muestras de *Jens Stoltenberg* hayan sido clasificadas correctamente un 94 % de las ocasiones pero, sin embargo, la tasa de aceptación sea baja respecto a la de los demás locutores. La gran mayoría de razones de verosimilitud de las muestras de los demás locutores respecto al modelo de *Jens Stoltenberg* tienen valores menores que -150. Decrementando el *threshold* a -150, la tasa de aceptación subiría de 0.78 a 0.87; eso sí, de las muestras del resto de locutores, se aceptarían como genuinas el 1.5 %. Con esto se quiere mostrar la posibilidad de establecer *thresholds*

independientes para cada uno de los locutores conocidos en función de los resultados obtenidos, adaptándolos según las necesidades del problema, ya que así puede mejorar el funcionamiento del sistema.

Es conveniente remarcar que, debido a la falta de datos (no hay varias muestras del mismo locutor diciendo la misma palabra o frase en cada una de ellas), esta implementación se ha probado como un sistema independiente de texto (Capítulo 2).

A pesar de ello, sería un sistema bastante fiable en cuanto a no aceptar a locutores no genuinos cuando estos son conocidos, ya que en *Tabla 3* se ha visto que, poniendo de umbral 0, solo fallaría en una muestra de las casi 500 que se han probado.

4.4. Implementación de un sistema automático de identificación con redes neuronales

Para este sistema de identificación, se han extraído las características de una muestra de audio mediante FFT y se han utilizado redes convolucionales para generar el modelo.

Para poner más a prueba a este sistema, las muestras de audio han sido mezcladas con distintos ruidos ambientales de forma aleatoria.

4.4.1. Librerías

Además de la libraría *Librosa* explicada anteriormente, se ha utilizado *TensorFlow* [21], que es una librería desarrollada por Google para la implementación de redes neuronales.

4.4.2. Código empleado

A continuación se muestra y explica el código utilizado.

Extracción de características

```
def audio_to_fft(audio):
    audio = tf.squeeze(audio, axis=-1)
    fft = tf.signal.fft(
        tf.cast(tf.complex(real=audio,
            imag=tf.zeros_like(audio)), tf.complex64)
    )
    fft = tf.expand_dims(fft, axis=-1)
    return tf.math.abs(fft[:, : (audio.shape[1] // 2), :])
```

Para la extracción de características, simplemente se ha obtenido la FFT de cada tensor correspondiente a cada muestra de audio.

Creación del modelo

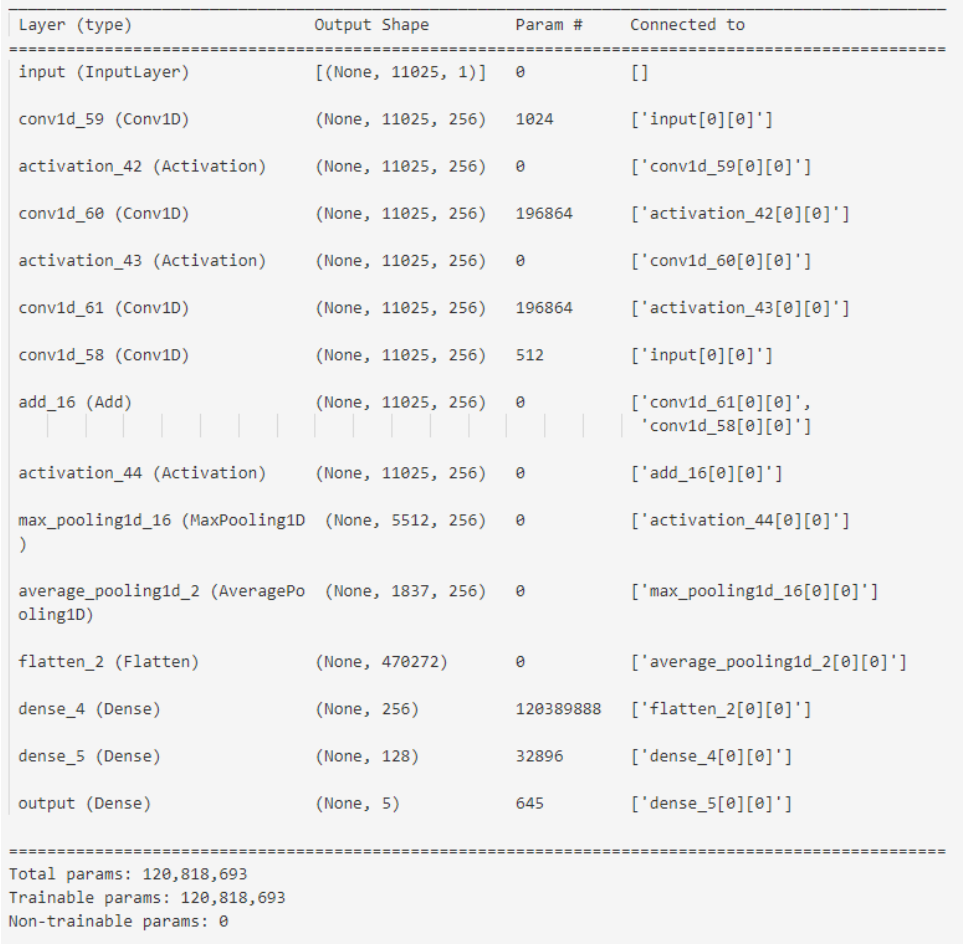
```
def block(x, filters):
    s = Conv1D(filters, 1, padding="same")(x)
    x = Conv1D(filters, 3, padding="same")(x)
    x = Activation("relu")(x)
    x = Conv1D(filters, 3, padding="same")(x)
    x = Activation("relu")(x)
    x = Conv1D(filters, 3, padding="same")(x)
    x = Add()([x, s])
    x = Activation("relu")(x)
    return MaxPool1D(pool_size=2, strides=2)(x)

def model(input_shape):
    inputs = Input(shape=input_shape, name="input")
    x = block(inputs, 256)
    x = AveragePooling1D(pool_size=3, strides=3)(x)
    x = Flatten()(x)
    x = Dense(256, activation="relu")(x)
    x = Dense(128, activation="relu")(x)
    outputs = Dense(5, activation="softmax", name="output")(x)
    return Model(inputs=inputs, outputs=outputs)
```

Para la creación del modelo, se han utilizado varias capas convolucionales de la API Keras [12]. Estas capas son unidimensionales y constan de 256 filtros,

y tienen de función de activación *relu*. Después, mediante *pooling*, se reduce la dimensionalidad de la entrada. Finalmente, se añaden tres capas más de activación, siendo la última de ellas *softmax*, para que se calcule la probabilidad de pertenencia a cada clase de cada muestra de audio.

A continuación se muestra una imagen con un resumen del modelo generado:



Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 11025, 1)]	0	[]
conv1d_59 (Conv1D)	(None, 11025, 256)	1024	['input[0][0]']
activation_42 (Activation)	(None, 11025, 256)	0	['conv1d_59[0][0]']
conv1d_60 (Conv1D)	(None, 11025, 256)	196864	['activation_42[0][0]']
activation_43 (Activation)	(None, 11025, 256)	0	['conv1d_60[0][0]']
conv1d_61 (Conv1D)	(None, 11025, 256)	196864	['activation_43[0][0]']
conv1d_58 (Conv1D)	(None, 11025, 256)	512	['input[0][0]']
add_16 (Add)	(None, 11025, 256)	0	['conv1d_61[0][0]', 'conv1d_58[0][0]']
activation_44 (Activation)	(None, 11025, 256)	0	['add_16[0][0]']
max_pooling1d_16 (MaxPooling1D)	(None, 5512, 256)	0	['activation_44[0][0]']
average_pooling1d_2 (AveragePooling1D)	(None, 1837, 256)	0	['max_pooling1d_16[0][0]']
flatten_2 (Flatten)	(None, 470272)	0	['average_pooling1d_2[0][0]']
dense_4 (Dense)	(None, 256)	120389888	['flatten_2[0][0]']
dense_5 (Dense)	(None, 128)	32896	['dense_4[0][0]']
output (Dense)	(None, 5)	645	['dense_5[0][0]']

Total params: 120,818,693
Trainable params: 120,818,693
Non-trainable params: 0

Figura 4.1: *Resumen del modelo de clasificación con redes neuronales*

Entrenamiento del modelo

```
history = model.fit(
    train_ds,
    epochs=4,
    validation_data=valid_ds,
)
```

Se utilizaron el 90 % de los datos para entrenar el modelo y el 10 % para ver la tasa de acierto ante muestras no conocidas. El entrenamiento se realizó en cuatro *epochs*, es decir, el modelo pasó un total de cuatro veces por el conjunto de datos durante el entrenamiento.

4.4.3. Resultados

En la siguiente tabla se recogen las tasas de acierto sobre el conjunto de prueba mostradas por pantalla durante el entrenamiento del modelo:

Epoch	**Porcentaje de acierto**
1	88.78 %
2	97.72 %
3	97.41 %
4	98.78 %

Tabla 4.4: *Resultados del sistema de clasificación con redes neuronales*

En ella, se puede ver que la tasa de acierto sobre el conjunto de prueba en la primera *epoch* era de más de un 92 % y, tras las cuatro *epochs*, de un 98.78 %. Esta última tasa de acierto mejora a la tasa de acierto del modelo GMM-UBM (97.60 %) en algo más de un 1 %; sin embargo, este modelo no funcionaría para verificación, ya que siempre clasifica una muestra de audio como uno de los cinco locutores con los que se ha entrenado (incluso clasificaría el ladrido de un perro como uno de ellos).

Capítulo 5

Conclusiones y trabajos futuros

El objetivo de este trabajo de fin de grado era realizar una revisión de diversas técnicas de verificación de locutor y explicar su funcionamiento teórico, además de realizar alguna implementación de sistemas tanto de verificación como de identificación para ver su relación. Se han visto técnicas estadísticas que permiten la implementación de estos sistemas.

La relación entre la verificación e identificación del locutor en el procesamiento del habla se basa en el objetivo común de determinar la autenticidad y la identidad de una persona a partir de su voz. ¿Es posible construir un sistema que sirva para las dos finalidades? Hemos visto que sí; cuando se verifica que un locutor es quien dice ser, se puede decir que se le está clasificando (identificando) como dicho locutor.

Sin embargo, cuando se identifica a un locutor, no se puede dar por verificado a ese locutor como el genuino; la identificación suele darse en un sistema cerrado de locutores y, si no hay un campo que sea *ninguno*, a ese locutor se le identificará con aquel con el que tenga más características en común.

Las posibilidades de este tipo de sistemas son numerosas: desde la ayuda que puede proporcionar en juicios y la creación de sistemas de seguridad por voz, hasta su aplicación en el ámbito de la medicina, ayudando a detectar enfermedades a través de la voz de los pacientes.

Un posible trabajo futuro a realizar sería estudiar e implementar algún sistema de verificación de locutor que funcione con redes neuronales y que utilice x-vectores o i-vectores en vez de MFCCs y GMM-UBM, ya que son técnicas que suelen obtener mejores resultados (un ejemplo de estudio es [10]).

Como ejemplo de aplicación en el ámbito médico, un grupo de investigadores en la Universidad de Nagoya en Japón ha realizado un estudio analizando características de habla de pacientes con Parkinson [1]. Si se recogieran más datos (muestras de voz) de personas sanas y personas con Parkinson, podría realizarse un análisis predictivo de si una persona presenta o no indicios de Parkinson con uno de estos sistemas, mejorándolo y adaptándolo para este problema.

Bibliografía

[1] ABC Salud (2023): *Los pacientes con párkinson hablan de forma diferente a los sanos.* [Enlace]

[2] Bäckström, T. (2019): *Cepstrum and MFCC.* [Enlace]

[3] Bäckström, T., Räsänen, O., Zewoudie, A., Pérez-Zarazaga, P., Koivusalo, L., Das, S., Gómez-Mellado, E., Bouafif-Mansali, M., Ramos, D. (2022): *Introduction to Speech Processing.* [Enlace]

[4] Biswas, A., Wennekes, E., Hussain-Laskar, R., Wieczorkowska, A. (2023): *Advances in Speech and Music Technology.*

[5] Brownlee, J., (2019): *A Gentle Introduction to Maximum a Posteriori (MAP) for Machine Learning.* [Enlace]

[6] Chapman, P., Clinton, J., Kerber, R., Khabaza, T., Reinartz, T., Shearer, C., Wirth, R. (2000): *CRISP-DM 1.0. Step-by-step data mining guide.*

[7] González-Rodríguez, J., Fiérrez-Aguilar, J., Ortega-García, J. (2003): *Forensic identification reporting using automatic speaker recognition systems.*

[8] Gündert, S. (2014): *Source code for melbank.* [Enlace]

[9] Kaggle (1-jun-2023): *Speaker Recognition Dataset: Prominent leaders speeches.* [Enlace]

[10] Kanagasundaram, A., Dean, D., Sridharan, S., Fookes, C. (2016): *DNN based Speaker Recognition on Short Utterances.*

[11] Kelly, F. (2014): *Automatic Recognition of Ageing Speakers.*

[12] Keras (2023): *Documentación de la API de Keras.* [Enlace]

[13] Librosa (2023): *Documentación de Librosa.* [Enlace]

[14] Mittal, A., Dua, M. (2021): *Automatic speaker verifcation systems and spoof detection techniques: review and analysis.*

[15] Morrison, G.S. (2013): *Tutorial on logistic-regression calibration and fusion: Converting a score to a likelihood ratio.*

[16] Morrison, G.S., Enzinger, E., Ramos, D., González-Rodríguez, J., Lozano-Díez, A. (2020): *Statistical models in forensic voice comparison.*.

[17] Reynolds, D.A., (2002): *An overview of automatic speaker recognition technology.*

[18] NumPy (2023). *Documentación de NumPy.* [Enlace]

[19] Scikit-learn (2023): *Documentación de Gaussian Mixture.* [Enlace]

[20] Snyder, D., García-Romero, D., Sell, G., Povey, D., Khudanpur, S. (2018): *X-Vectors: Robust DNN Embeddings for speaker recognition.*

[21] TensorFlow (1-jun-2023): *Documentación de TensorFlow.* [Enlace]

[22] Wikipedia (20-jul-2022): *Ventana (función).* [Enlace]

[23] Wikipedia (2023) *Escala Mel.* [Enlace]

[24] Wikipedia (2023): *Filter bank.* [Enlace]

[25] Wikipedia (2023): *Maximum a posteriori estimation.* [Enlace]