DISEÑO DE SOFTWARE IDENTIFICADOR DE SEÑALES MUSICALES

Luca Blasi[1], Débora Reyes[2] y Agustín Sosa Lüchter[3]

[1]Universidad Nacional de Tres de Febrero
lucajblasi@gmail.com
[2]Universidad Nacional de Tres de Febrero
debora.reyes@hotmail.com
[3]Universidad Nacional de Tres de Febrero
agustinsosaluchter@gmail.com

Resumen — *Se diseña un motor de búsqueda de audio, que puede identificar una canción incluída en una pequeña base de datos a partir de una grabación realizada con una notebook o un celular. El código utiliza un algoritmo de identificación de picos de amplitud, aplicando métodos para condensar la información y mejorar el rendimiento. Se obtiene las huellas digitales (fingerprints) de los archivos de la base de datos, y luego comparándolas con las huellas digitales de la grabación realizada. Debido a la utilización de picos de amplitud para realizar la identificación, la implementación del código es efectiva aún en situaciones de ruido, y es mayormente independiente de la calidad del micrófono del celular o de la notebook utilizada.*

1. INTRODUCCIÓN

En este trabajo práctico se analizan el diseño e implementación de un algoritmo que permita identificar señales musicales de una base de datos. La idea es grabar señales de este tipo en situaciones cotidianas mediante instrumental común como un celular o notebook para que el software indique si la grabación corresponde a alguna canción de la base de datos, y en caso de ser afirmativo, a cuál de ellas.

Este algoritmo debe sobreponerse a varias dificultades. Para empezar debe ser capaz de identificar la canción a través de una grabación realizada con un micrófono de un celular o notebook. Estos micrófonos en general están optimizados para la voz humana, por lo que poseen una respuesta en frecuencia y un rango dinámico acotados. Por otro lado la señal musical grabada se encuentra mezclada con ruido de fondo, posiblemente afectada por campo reverberante y depende del sistema en el que se reproduzca.

Para resolver esos problemas, se realiza un proceso de *acoustic fingerprinting* (o extracción de huella digital acústica). Las fingerprints se pueden detectar en el dominio temporal y en el dominio frecuencial, y convierte la señal de audio en una secuencia de las características más relevantes de este. A su vez, cada fingerprint asociada a un archivo de la base de datos es única.

Estas características las convierten en una buena herramienta para determinar la equivalencia de dos señales musicales, ya que no compara las dos señales propiamente dichas (que usualmente son de gran tamaño) sino que compara las fingerprints asociadas (mucho más pequeñas relativamente). La utilización de fingerprints requiere menos memoria y capacidad de procesamiento ya que los datos procesados son de menor tamaño, y también permite una búsqueda más eficiente a través de la base de datos, por las mismas razones. Por último, la comparación también es eficiente ya que a través del proceso de extracción de fingerprints se elimina de ellas las variaciones perceptuales, que son irrelevantes para la comparación.

2. MARCO TEÓRICO

2.1. Espectrogramas

La comparación de picos de espectrogramas (tiempo-frecuencia) es una buena manera de hacer coincidir dos señales en presencia de distorsión. Los picos se determinan como puntos con la mayor energía de una región centrada alrededor de ellos. Una vez determinados, un espectrograma complejo puede reducirse a una constelación de coordenadas, en donde la componente de amplitud se elimina.

Este mecanismo tiene una ventaja muy importante. Es básicamente inmune a cambios en la ecualización de la señal, por lo que aún en presencia de ruido y otras variables, el patrón de puntos debería ser similar para dos muestras de la misma canción. Sin embargo, puede ocurrir que en el mapa de constelaciones de la muestra grabada se encuentren picos introducidos por la distorsión, pero el número de coincidencias será suficiente como para poder realizar la comparación.

2.2. Funciones Hash

Una huella digital acústica (o audio fingerprint) puede percibirse como un resumen"de una señal de audio. Para extraer las fingerprints se utilizan las funciones hash, empleadas habitualmente en la criptografía. Las funciones hash tienen como entrada un número elevado de elementos que suelen ser cadenas (X), y los convierte en un rango de salida finito, normalmente cadenas de longitud fija.

Entonces, las funciones hash permiten comparar dos objetos *X* e *Y* de una manera mucho más eficiente, con mayor rapidez y menor costo de procesamiento, comparando sus respectivos valores de hash *H(X)* e *H(Y)*. La igualdad de las funciones hash implica la igualdad de los conjuntos, con una probabilidad de error muy pequeña. Para una función hash bien diseñada, la probabilidad de error es 2^n, donde n es igual al número de bits del valor de la función hash.

No obstante, las funciones hash por sí solas no representan la solución al problema. Una misma señal musical en formato *WAV* y en formato *MP3 128Kb/s* presenta dos formas de onda totalmente diferentes, aunque la diferencia sea imperceptible al oído humano. Por lo tanto, las funciones hash de ambos archivos serán también distintas. Por supuesto, las funciones hash de un archivo de una base de datos y de una grabación de aire también serán completamente diferentes.

2.3. Fingerprints

Las acoustic fingerprints son un identificador para archivos de audio basados en el contenido del archivo. Con ellas se puede identificar un patrón o "firma" de un archivo de audio, para que este pueda ser reconocido desde una base de datos, sin necesidad de disponer de información (meta data) acerca de este.

Su uso se basa en la detección de una muestra de audio y posterior envío a una base de datos, para buscar coincidencias en esta y devolver información acerca de la muestra analizada. Estas tienen la intención de capturar las características más relevantes de la percepción de audio para cumplir con este propósito.

La extracción y comparación de fingerprints debe ser rápida y fácil. Dado un fragmento de audio y su correspondiente huella dactilar, se busca la vía más rápida para encontrar su mejor coincidencia- Se busca también maximizar la eficiencia, tomando en cuenta desde el número de iteraciones en la búsqueda hasta la longitud de la muestra, ya que dependiendo de estas el procesamiento puede resultar computacionalmente costoso.

Existen cinco parámetros básicos que definen a las fingerprints:

- **Robustez:** Cuantifica la efectividad para identificar señales severamente degradadas. La taza de falsos negativos es una manera de expresarla.
- **Confiabilidad:** Cuantifica la efectividad en general. La taza de falsos positivos es una manera de expresarla.
- **Tamaño:** Cuantifica qué tanta capacidad de almacenamiento se necesita para una fingerprint. Dado que usualmente se almacenan en la memoria RAM, se expresa en bits por segundo.
- **Sensibilidad:** Cuantifica cuántos segundos de audio se necesitan para realizar una identificación.
- **Escalabilidad:** Cuantifica cuánto se tarda en encontrar una fingerprint en la base de datos de fingerprints. En los sistemas comerciales, se espera que este tiempo sea en el orden de los milisegundos.

Estos parámetros básicos se encuentran íntimamente relacionados entre sí ya que la tasa de falsos positivos está inversamente relacionada con el tamaño de la fingerprint. Por ende a modo de ejemplo, si se busca menor sensibilidad para una identificación más rápida, se debe extraer una fingerprint más grande para mantener el nivel de confiabilidad.

3. PROCEDIMIENTO

La programación del código se realiza en el lenguaje Python, y se utilizan múltiples librerías en su implementación. El mismo sigue el siguiente lineamiento:

- Se obtienen los picos del espectrograma.
- Combinatorial Hashing de los picos generados.
- Elimina hashes duplicados y convierte de lista a dict.
- Se crea base de datos.
- Compara el fingerprint del sample contra los fingerprints que se encuentran almacenados en database.pkl. Decide si existe un match.

3.1. Obtención de picos del espectrograma

En primer lugar se importa el archivo de audio a través de *SciPy* obteniendo su frecuencia de muestreo y sus datos en una variable de tipo array. Luego se calcula la Transformada de Fourier de Tiempo Corto (*STFT*) del array. Esto da como resultado una matriz compleja con información de tiempo y frecuencia. Para analizar la magnitud de la *STFT* se calcula el valor absoluto de la matriz, obtiendo así la amplitud para cada par de tiempo-frecuencia. También se pasa la magnitud a dB.

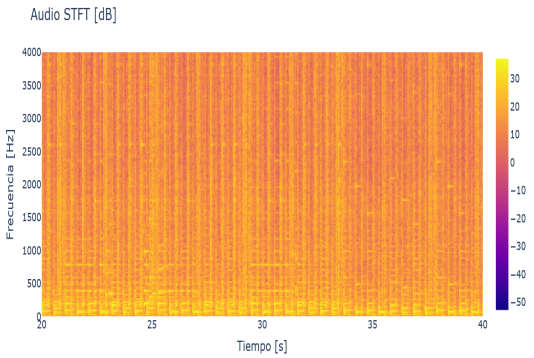

Figura 1: Ejemplo de STFT

Para realizar la comparación, se debe eliminar el parámetro de amplitud y analizar solamente los pares de tiempo-frecuencia en los puntos que tienen amplitud máxima. Para ello se genera una matriz que define el área alrededor de un punto a analizar y se buscan los máximos locales. Luego se generan dos *array* que contienen los valores de tiempo y frecuencia para los cuales la amplitud es máxima.

Finalmente se genera una lista donde cada índice contiene un par de tiempo-frecuencia. Es en esta instancia que se genera el *mapa de constelaciones* que servirá para realizar la comparación de las señales. Se determina la amplitud de los picos, de modo de poder establecer un mínimo que sirva como *compuerta*. Luego se filtran esos picos.

3.2. Combinatorial Hashing

Se define una función *hasher* que realiza el ***hashing*** de los picos generados. Para ello, primero toma un par de picos *i* y *j*. Toma el tiempo entre ambos y luego realiza la resta entre ellos, obteniendo un *dt*. También se obtienen las frecuencias de ambos picos, denominadas *f1* y *f2*. A través del módulo *hashlib* se combina la información de *f1*,*f2* y *dt* en un hash. Un pico *i* se combina con varios picos *j*, y así para todos los picos.

Por último se procede a eliminar los hashes duplicados y convierte la lista en dict. La ventaja del tipo dict es que es más facil y considerablemente más rápido buscar una concordancia de los hashes entre dos fingerprints de dos audios separados (muestra y base de datos).

Esta acción, en la teoría, empeora la capacidad de encontrar coincidencias ya que se reduce el número de tuplas para efectuar la comparación. Por otro lado, eliminar los duplicados facilita las operaciones ya que garantiza que toda coincidencia de hashes entre la muestra grabada y el archivo original tenga una sola diferencia temporal. A su vez, en la práctica, la pérdida de tuplas es mínima y no tiene un efecto perceptible en el rendimiento. Un detalle es convertir la lista de hashes en un diccionario, la ventaja de este tipo de variable es que es más fácil y considerablemente más rápido buscar los hashes en comun entre dos fingerprints de audios separados.

3.3. Creación de base de datos

Para la creación de la base de datos que se usará para identificar las señales, se crea una función que importa los audios ubicados en una carpeta y aplica a cada uno de ellos el proceso descripto en el punto anterior. Se guardan los fingerptint extraidos de cada uno en una lista y finalmente a través de la libreria pickle se guardan las variables en un archivo de tipo .pkl.

Además se crea otra función que permite actualizar esta base de datos según la ubicación de una carpeta. Evita procesar archivos de los que ya se obtuvo su fingerprint y elimina los fingerprints de archivos borrados.

3.4. Comparación de fingerprints

En primer lugar se ejecuta una función que permite grabar un audio en tiempo real de duración definida por el usuario por medio del método *sounddevice*, o también permite elegir un archivo de audio ubicado en cualquier carpeta. Luego se extrae el fingerprint de esta muestra.

Una nueva función se encarga de comparar el fingerprint de la muestra con los fingerprints que se encuentran almacenados en la base de datos. A través de un bucle *for* se compara el hash de la muestra con los hashes de las canciones de la base de datos, obteniendo intersecciones. Luego se crean dos nuevas lista, que contienen el tiempo inicial y el hash encontrado en la intersección, cuyos elementos tienen la forma *(t1,hash)*. Una lista corresponde a la muestra grabada y la otra a la canción de la base de datos. Esto es así porque los *t1* de la muestra y de la canción son diferentes.

Esa diferencia se utiliza para crear una tercer lista, que contiene las diferencias de los *t1* de la canción y de la muestra para un mismo hash. El histograma que se crea coloca los elementos de *dt* según su valor en 100 'barras' diferentes. Entonces cuantos más *dt* tengan el mismo o similar valor, mayor va a ser el valor de la barra correspondiente.

Por último, se encuentra el máximo significativo. Se busca un valor máximo del histograma, ya que de existir

una coincidencia, estará representada por ese valor. Esto es, para que no se produzcan grandes desvíos al momento de sacar promedios y calcular la desviación estándar. También se fija la condición de que este máximo sea superior a un valor mínimo de 60 coincidencias para evitar reiterados falsos positivos. Se calculan empleando la librería *pandas*. Luego se determina un umbral dado por el promedio y la desviación estándar, en la forma $avg + 2 * std$. Se opta por este umbral ya que en una distribución normal contiene al 95 % de los casos. Entonces, si un valor se excede de este umbral, se trata de un *outlier*. Por otro lado, si se encuentra dentro del mismo, se asume que hay una concordancia y se muestra el título de la canción correspondiente.

Se crea una interfaz gráfica de usuario para hacer la utilización de programa más interactiva. *Open file* y *Record audio* se encargan de la adquisición de datos, ya sea un archivo de audio o una grabación, respectivamente. *Recording length* permite elegir la duración de la grabación que se desea efectuar. *Create database* crea un fingerprint de la base de datos a base de los archivos contenidos en la carpeta audio_database. *Update database* actualiza el fingerprint de la base de datos. Evita procesar archivos que ya tienen fingerprint, y elimina fingerprints de archivos borrados. *Run* compara los fingerprints y devuelve si existe coincidencia o no con la base de datos.

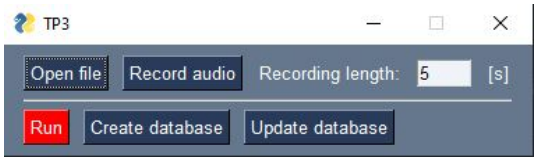

Figura 2: Interfaz de usuario

4. RESULTADOS Y ANÁLISIS

Luego de que todos los hashes de las muestras se hayan utilizado para buscar pares en la base de datos, se observan los gráficos resultantes. Debido a que los hashes son altamente específicos, los scatterplots poseen valores muy dispersos. En la figura siguiente se ilustra un histograma y un scatterplot en donde la señal de la base de datos no concuerda con la señal grabada.

Figura 3: Histograma y scatterplot de los offset de diferencias temporales, en este caso sin concordancia

Para obtener estos gráficos se compara deliberadamente una grabación de una canción con otra de la base de datos. Se observan algunas asociaciones fortuitas, pero sin embargo no se aprecia una correspondencia lineal. Cuando efectivamente se produce un *match*, se observa que en el scatterplot las locaciones de los hashes adquieren la forma de una recta como se aprecia en la siguiente figura.

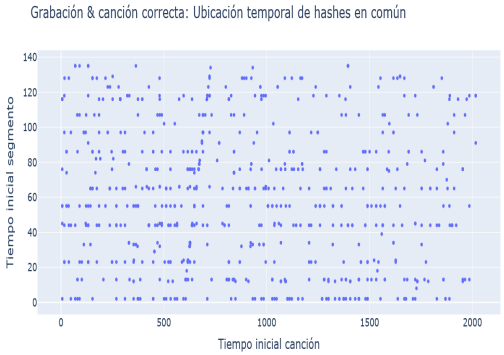

Figura 4: Scatterplot de las concordancias de las locaciones de hashes, de una señal grabada

Se puede observar en la siguiente figura, cómo esa misma linealidad se observa al graficarse el histograma con las diferencias temporales.

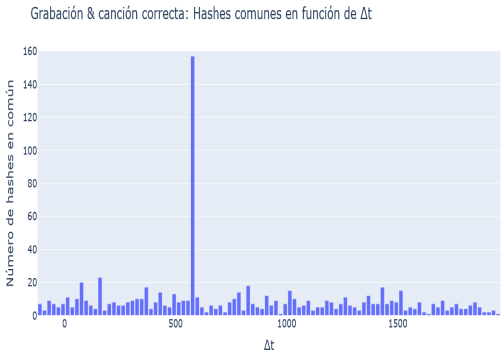

Figura 5: Histograma de diferencias temporales

Cuando se utiliza el propio segmento de la base de datos como muestra para realizar el análisis (en lugar de una grabación) la recta por supuesto se observa con mayor claridad.

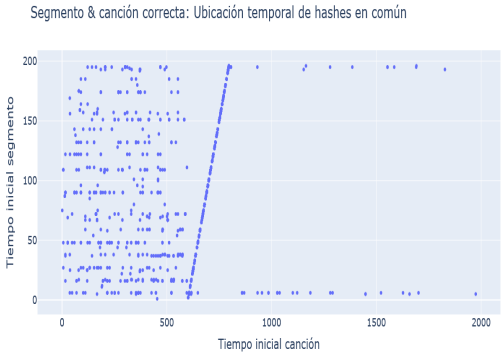

Figura 6: Scatterplot de las concordancias de las locaciones de hashes, de un segmento de una canción de la base de datos.

Luego se ponen a prueba los límites del algoritmo al modificar las condiciones de los experimentos. A las señales grabadas se les agrega digitalmente reverberación y ruido y el algoritmo logra emparentar las fingerprints sin problemas. Sin embargo, cuando se realizan cambios en el pitch y en la velocidad de reproducción (o ambos) de la señal musical, el código falla en asociarla a un archivo de la base de datos. Esto es esperable ya que el procedimiento se basa en buscar coincidencias en las frecuencias pico, y estas son alteradas al modificarse el pitch y la velocidad de reproducción.

Respecto al tiempo de procesamiento, resulta ser lo suficientemente rápido ya que el tiempo en encontrar una coincidencia varía entre 0,2 s y 0,8 s. Hay que destacar que la base de datos es muy pequeña, por lo que se podría analizar qué sucede con este tiempo a medida que se aumenta la misma.

5. CONCLUSIONES

A través de este código se logra obtener satisfactoriamente las fingerprints de los archivos de la base de datos y de la grabación, y al efectuar la comparación se obtiene el resultado esperado. Pese a la previsible falla al introducir cambios de pitch, el programa funciona adecuadamente incluso cuando es sometido a grandes cantidades de reverberación, distorsión y ruido de fondo. La corrección de esta limitación, y la creación de una interfaz gráfica interactiva con mayores prestaciones, quedan como trabajos futuros.

6. REFERENCIAS

[1] Haitsma, Jaap & Kalker, Ton. (2002). A Highly Robust Audio Fingerprinting System. Proc Int Symp Music Info Retrieval. 32.

[2] Wang, Avery. (2003). An Industrial Strength Audio Search Algorithm.

[3] https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/

[4] https://github.com/worldveil/dejavu

[5] Stinson, D.R.. (1994). Combinatorial techniques for universal hashing. Journal of Computer and System Sciences. 48. 337–346. 10.1016/S0022-0000(05)80007-8.