Universidad del Rosario | Escuela de Ingeniería, Ciencia y Tecnología | MACC Matemáticas Aplicadas y Ciencias de la Computación

Informe - Proyecto Shazam

Laura Sofía Ortiz

Mayo de 2023

Este proyecto se divide en dos partes, las cuáles se explicaran a detalle.

1. Lab Procedure - Part I

Para esta parte, se va a crear el sistema de entrenamiento completo para Shazam, que extrae las huellas dactilares de todos los archivos MP3 en una carpeta designada y crea una base de datos. Puede usar cualquier archivo MP3 que desee. Los principales pasos de este procedimiento son los siguientes:

1. Leer la canción usando mp3read.
2. Promediar los dos canales, restar la media y reducir la muestra.
3. Calcular el espectrograma de la canción usando la función spectrogram.
4. Encuentrar los picos locales del espectrograma usando circshift en un bucle.
5. Crear un umbral del resultado del paso anterior, para terminar con una tasa específica de picos retenidos por segundo de sonido.
6. Encuentrar pares de picos proximales y agréguelos, y cárguelos en una tabla hash.

Estos pasos se explican en detalle a continuación.

1.1. Reading an MP3 file: mp3read

Para leer un archivo .mp3 en Matlab (las canciones) se utilizó la función mp3read diseñada por el profesor Dan Ellis, Universidad de Columbia.

```
folder_name = '/home/laura/Desktop/univ/univ2023-1/senales/Proyecto1/parte1/parte2/Canciones';
songs = getMp3List(folder_name);
display(songs)
```

Figura 1: Comando para leer las canciones .mp3 del folder.

1.2. Fingerprint

Para realizar los numerales del 2 al 5, se creó una función ***fingerprint.m***, la cuál toma como argumentos una señal de sonido y su frecuencia de muestreo. Esta función toma una señal de audio y crea una huella digital de la misma. La huella es una representación simplificada de la señal que se puede utilizar para identificar la señal en el futuro.

Lo primero que hace la función es re-muestrear la señal a una tasa de muestreo más baja, y calcular el espectrograma de la señal, el cuál es una representación visual de la frecuencia en función del tiempo de la señal. En esta primera parte del preprocesamiento nos hacen las siguientes preguntas:

¿Por qué podría ser una buena idea volver a centrar nuestra señal?, volver a centrar la señal implica restar el valor medio de la señal de la señal misma. Este es un paso de preprocesamiento común en el procesamiento de señales porque elimina el componente de DC de la señal, que es el que tiene una frecuencia de cero Hz. Quitar el componente facilita ver los otros componentes de frecuencia de la señal y puede ayudar a prevenir la saturación de ciertas partes de la tubería de procesamiento de la señal.

Es común estudiar visualmente el registro de la magnitud del espectrograma. ¿Por qué podría ser esto una buena idea?, estudiar el logaritmo de la magnitud del espectrograma es común en el procesamiento de señales porque permite visualizar mejor el rango de valores presentes en el espectrograma. Esto facilita la identificación y comparación de diferentes componentes de frecuencia en el espectrograma, especialmente cuando varían mucho en magnitud.

Luego de realizar lo anterior, se buscan los picos en el espectrograma. Esto se hace mediante la comparación de cada punto del espectrograma con sus vecinos inmediatos en una ventana de un tamaño específico en todas las direcciones. Si el punto es mayor que ambos vecinos, se considera un pico. Este proceso se repite para cada punto, produciendo una matriz de picos. Para esta parte se nos hace la siguiente pregunta:

¿Crees que tomar huellas dactilares de la canción usando picos proporciona alguna ventaja inherente sobre el uso de valles?, tomar huellas dactilares de una canción usando picos proporciona algunas ventajas inherentes sobre el uso de valles. Los picos tienden a ser más robustos y fáciles de identificar en comparación con los valles, especialmente en entornos ruidosos. Además, es más probable que los picos en el espectrograma se correspondan con características distintas e identificables de la señal de audio, que a menudo son más importantes para la toma de huellas dactilares que las regiones entre ellos.

La función aplica un umbral a los picos locales para eliminar los picos que no son significativos. La cantidad de picos que se mantienen después del umbral depende de la cantidad de picos que se desean mantener por segundo (30 en este caso). Para esta parte del umbral nos hacen las siguientes preguntas:

¿La distribución de los picos es uniforme?. La distribución de picos en un espectrograma no suele ser uniforme. Tienden a agruparse alrededor de regiones de alta energía en la señal. Según la naturaleza del sonido, pueden estar muy juntos o más dispersos.

¿Los picos están muy juntos? Si es así, ¿es esto algo bueno?. Tener picos muy juntos puede ser tanto bueno como malo. Por un lado, puede proporcionar más información sobre la señal, facilitando la distinción entre diferentes sonidos. Por otro lado, también puede generar problemas con la fuga espectral y el ruido, lo que puede dificultar la identificación precisa de los picos.

Finalmente, la función realiza una gráfica del espectrograma, una de los picos locales, y otra con los picos después del umbral. A continuación podemos observar un ejemplo de las gráficas de una canción:

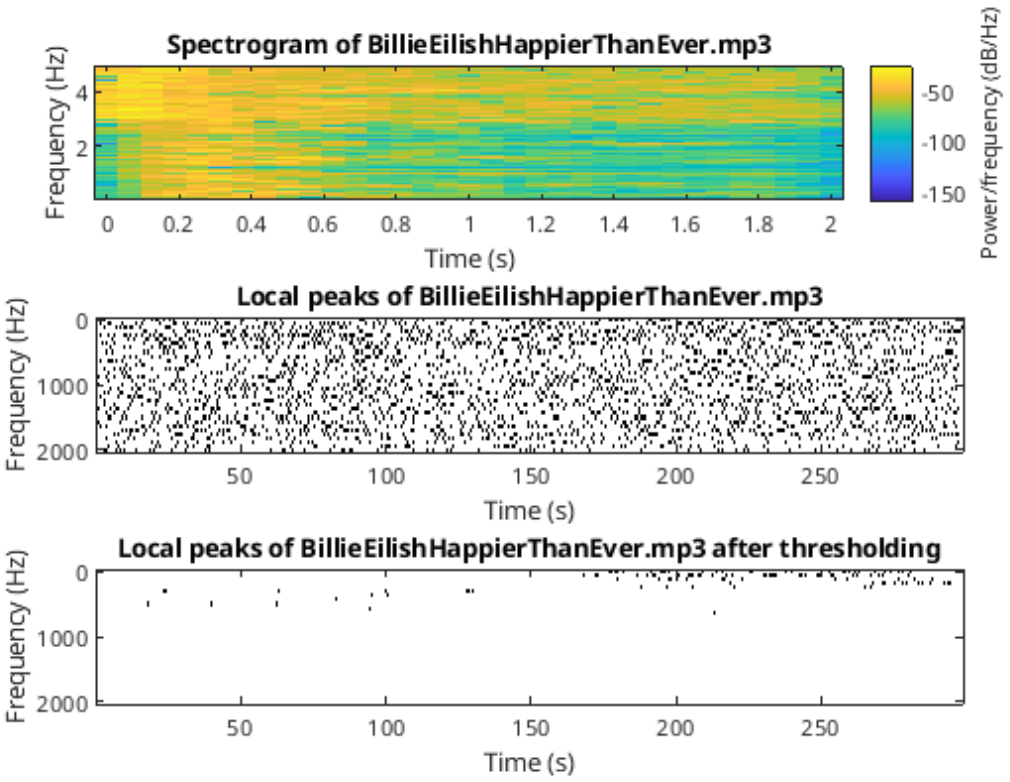

Figura 2: Gráficas para la canción Happier than Ever.

1.3. Find peak pairs

Para realizar el numeral 6, la parte de encontrar los pares de picos, se nos dió una función ***convert_to_pairs.m*** que toma una matriz de picos y devuelve una tabla de pares que están cerca tanto en tiempo como en frecuencia.

El código encuentra pares considerando cada pico y buscando otros picos dentro de una ventana designada ubicada en relación con él. Durante la búsqueda, limitamos el número de pares que aceptamos por el parámetro fanout (número máximo de pares por pico). El código escanea la ventana columna por columna, aceptando los primeros emparejamientos que encuentra.

Finalmente, la función devuelve la lista de pares de picos como una matriz, que tiene cuatro columnas que representan los índices de tiempo y frecuencia de los dos picos en cada par. Los pares que no se completaron se descartan antes de devolver la lista.

1.4. Train database

Por último, para concluir el numeral 6 de la primera parte del proyecto, se nos proporcina dos plantillas de códigos. El primero, ***add_to_table.m***, que edita una variable global llamada hashtable, pero discutiremos este código más en la parte 2 del proyecto. Y el otro código, ***Parte1_Shazam.m***, que es un script que busca todos los archivos MP3 en una carpeta designada y los procesa si aún no están en la base de datos.

Para cada canción en la lista, el código verifica si la canción ya existe en la base de datos. Si la canción no se encuentra en la base de datos, se agrega la huella dactilar y el nombre de la canción a la base de datos. Para agregar la huella dactilar, el código utiliza las funciones de los pasos

anteriores, ***fingerprint*** y ***convert_to_pairs*** para generar una lista de pares de huellas dactilares, que se agregan a la tabla hash mediante la función ***add_to_table*** (el valor hash se calcula a partir del vector $(f1, f2, t2 - t1)$, de modo que los pares de picos con las mismas frecuencias y separación en el tiempo se consideran una coincidencia).

Finalmente, si se han agregado nuevas canciones a la base de datos, se guardan los nombres de las canciones y la tabla hash en archivos separados llamados, SONGID2.mat y HASHTABLE2.mat, respectivamente.

2. Lab Procedure - Part II

En esta última parte, vamos a identificar un segmento de música, usando la base de datos que entrenamos en la primera parte. Los pasos principales del algoritmo son los siguientes:

1. Cargar HASHTABLE2.mat y SONGID2.mat que fueron creados por ***Parte1_Shazam.m*** en la parte 1.
2. Preparar un clip de música para la identificación.
3. Extraer la lista de pares de frecuencias del clip.
4. Buscar las coincidencias en la tabla hash, calcular las compensaciones de tiempo y ordenarlas por canción.
5. Identificar la canción con la mayor cantidad de coincidencias para un solo desplazamiento de tiempo consistente.

A continuación lo discutimos con más detalle.

2.1. Match clip to song

La coincidencia de canciones se realizará con la función ***match_segment.m***, para la cuál nos dieron una plantilla. Esta función acepta un segmento de sonido y una frecuencia de muestreo como argumentos y emite la canción que mejor se adapta, así como un nivel de confianza (que por defecto es 1). Las variables hashtable y songid deben existir como variables globales para que esta función funcione correctamente.

En específico, la función se encarga de obtener los picos del clip a través de la función ***fingerprint***, la cuál devuelve un vector con las posiciones de los picos en el clip. Además, se construye una celda de coincidencias para cada canción de la base de datos. Para cada par de picos en el clip, se calcula un hash simple, con la funcion ***simple_hash*** la cuál se utiliza para calcular un índice hash único para un conjunto de valores f1, f2 y deltaT(t2 - t1), para poder buscar y comparar las huellas digitales de las canciones en una tabla hash. Para esta parte se nos hace la siguiente pregunta:

¿Por qué nos preocupamos por las compensaciones en lugar del vector de tiempo en sí?. En la toma de huellas dactilares de audio, estamos interesados en identificar un clip de audio en particular, independientemente de su hora de inicio dentro de una grabación de audio más larga. Por lo tanto, no nos importa la información de tiempo del clip de audio, sino las compensaciones entre los picos identificados u otras características de audio que usamos para la identificación. Al usar las compensaciones, podemos hacer coincidir clips de audio que pueden tener tiempos de inicio diferentes, pero que comparten patrones similares de picos o características.

Luego de calcular el hash simple, este se utiliza para buscar coincidencias en la tabla hash, que es una matriz de dos columnas, la primera columna contiene los índices de las canciones que han

sido asignadas a cada hash, y la segunda columna contiene los tiempos (en muestras) de los picos que coinciden con ese hash. Si se encuentra una coincidencia en la tabla hash, la función agrega las coincidencias para cada canción en la celda de coincidencias correspondiente.

Finalmente, se encuentra el valor más frecuente de los tiempos de coincidencia en cada celda de coincidencias para cada canción en la base de datos, y se devuelve el índice de la canción con el conteo del modo más alto como la mejor coincidencia (bestMatchID).

```
% Song decision and confidence
[~, bestMatchID] = max(counts);
confidence = 1;
```

Figura 3: Comando para calcular la mejor coincidencia de la canción.

2.2. Test Shazam

Ya para lo útimo, se nos dió otra plantilla que llame como ***Parte2_Shazam.m***, para probar y usar el sistema Shazam. Primero, se cargan las variables hashtable y songid que se guardaron durante el proceso de entrenamiento, si aún no están en el espacio de trabajo. Entonces esta función proporciona dos opciones, seleccionar un segmento aleatorio de una canción del conjunto de entrenamiento o grabar el sonido de una canción por medio del micrófono.

Finalmente, se utiliza la función ***match_segment*** para hacer coincidir la canción, ya sea si se selecciona un segmento aleatorio o si se utiliza el micrófono. Y para esta parte se realiza la siguiente pregunta:

¿Cuál es la precisión de su programa usando las siguientes pruebas?. La opción del micrófono, no funciona al 100 %, tal vez porque estoy ubicando mal el celular para que escuche la canción o quizas por otro motivo. Por otro lado, si se escoge la opción de elegir un segmento aleatorio de una canción de la base de datos, la mayoría de pruebas coincidieron bien, podemos ver un ejemplo a continuación:

Figura 4: Prueba de la coincidencia de canciones

Nota: Para probar el código se debe correr el archivo ***Parte2_Shazam***.