Tarea 2

CC5213 – Recuperación de Información Multimedia
Profesor: Juan Manuel Barrios
Fecha de entrega: 04 de junio de 2025

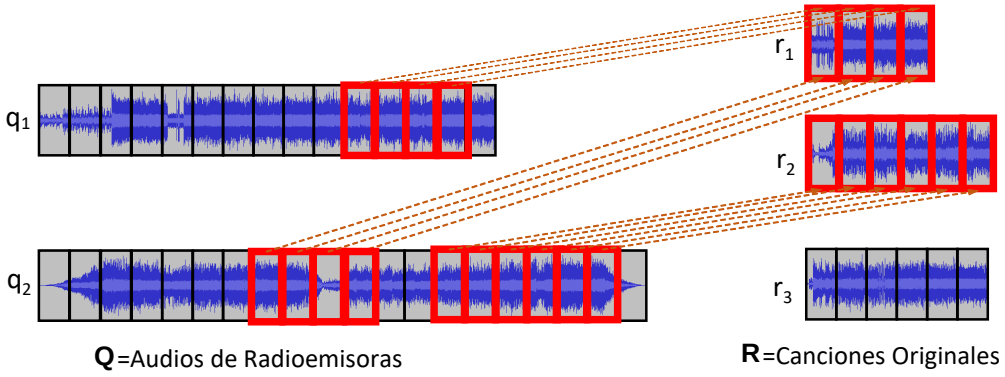
El objetivo de esta tarea es implementar un **detector de canciones emitidas por radioemisoras**. Dado un conjunto de canciones originales **R** y un conjunto **Q** con grabaciones de emisoras de radio, se desea determinar para cada archivo de audio de **Q** todas las apariciones de las canciones de **R**.

Cada audio de **Q** corresponde a la programación emitida por una radioemisora durante 1 hora. Cada audio de **R** corresponde a un trozo de canción, que puede tener un largo entre los 5 y los 40 segundos. Cada audio de **Q** contiene del orden de 20 a 40 emisiones de canciones de **R** y el resto del tiempo contiene otras canciones que no están en **R**. Puede asumir que cada emisión de **R** siempre es en su totalidad, es decir, si un audio de **R** tiene un largo de 15 segundos, cada vez que aparezca en un audio de **Q** aparecerán los 15 segundos. Cada audio de **R** puede aparecer cero, una o más veces en cada audio de **Q**.

Cuando es emitida una canción de **R**, podría tener alguna modificación en su calidad. Específicamente, en cada emisión una canción de **R** podría aparecer: de igual calidad (sin modificación), con menor volumen, con mayor volumen, con baja calidad de audio, o mezclado con ruido de fondo.

La tarea consiste en implementar un software que recibe los conjuntos **Q** y **R** (carpetas con archivos de audio) y genera un archivo de texto señalando para cada audio $q \in$ **Q** una lista con las emisiones detectadas. Cada emisión se define por el segmento de tiempo de q (tiempo de inicio y largo en segundos) y el nombre de la canción de **R** que se escucha en ese segmento.

Figura 1: Cada ventana de audio en **Q** se asocia con la ventana más parecida en **R**



Para la tarea deberá implementar tres comandos:

1. **Calcular descriptores de audio:** Un comando que recibe el nombre de una carpeta que contiene archivos de audio (.m4a) y el nombre de una carpeta donde se guardarán los descriptores de los archivos de audio:

```
python tarea2-parte1.py [carpeta_entrada_audios]
                        [carpeta_salida_descriptores]
```

El formato, el contenido y la cantidad de archivos a guardar en la carpeta de salida es a su elección.

Se recomienda dividir cada audio en ventanas de largo fijo y calcular un descriptor MFCC por ventana. Guarde los descriptores de todos los audios en uno o más archivos en la carpeta de salida, incluyendo también datos de la ventana que representa cada descriptor.

Si lo necesita, puede usar ffmpeg para convertir de formato de los audio, use la carpeta de descriptores de salida para guardar archivos temporales.

2. **Fase comparación de descriptores:** Un comando que recibe el nombre de la carpeta donde están los descriptores de audios de radio **Q**, la carpeta donde están los descriptores de las canciones **R** y el nombre de una carpeta de salida donde se guardarán los resultados de la comparación de ventanas:

```
python tarea2-parte2.py [carpeta_descriptores_radio_Q]
                        [carpeta_descritores_canciones_R]
                        [carpeta_salida_ventanas_similares]
```

Lee los descriptores de los audios de radio **Q** y los descriptores de las canciones **R** (ambos generados por el comando 1), para cada ventana de **Q** determina la o las ventanas más similares de **R** y escribe en la carpeta de salida uno o más archivos con los datos de las ventanas similares.

El formato y contenido de lo que guarde en la carpeta de salida es a su elección. Se recomienda incluir, al menos, el tiempo de inicio de cada ventana de **Q** junto con el tiempo de inicio de la ventana más similar de **R**. La Figura 2 muestra un posible archivo guardado en la carpeta de salida.

Figura 2: Posible archivo generado por `tarea2-parte2.py`.
Para cada ventana de **Q** (archivo y tiempo de inicio de la ventana) muestra la ventana de **R** más cercana (archivo y tiempo de inicio)

radio_1.m4a	428.0	cancion_05.m4a	23.0
radio_1.m4a	**429.0**	**cancion_03.m4a**	**6.0**
radio_1.m4a	**430.0**	**cancion_03.m4a**	**7.0**
radio_1.m4a	**431.0**	**cancion_03.m4a**	**8.0**
radio_1.m4a	432.0	cancion_07.m4a	20.0
radio_1.m4a	433.0	cancion_08.m4a	3.0
radio_1.m4a	**434.0**	**cancion_03.m4a**	**11.0**
radio_1.m4a	435.0	cancion_07.m4a	20.0
radio_1.m4a	**436.0**	**cancion_03.m4a**	**13.0**
radio_1.m4a	**437.0**	**cancion_03.m4a**	**14.0**
radio_1.m4a	438.0	cancion_02.m4a	3.0
radio_1.m4a	**439.0**	**cancion_03.m4a**	**16.0**
radio_1.m4a	440.0	cancion_05.m4a	22.0

Opcionalmente, podría incluir información extra como la distancia a la que está el más cercano, o información del segundo vecino más cercano.

3. **Fase detección de secuencias similares:** Un comando que recibe el nombre de la carpeta donde están las ventanas similares entre **Q** y **R** y el nombre de un archivo de salida con las canciones detectadas:

```
python tarea2-parte3.py [carpeta_ventanas_similares]
                        [archivo_salida_detecciones_txt]
```

Lee el archivo de ventanas similares (generado por el comando 2), identifica secuencias de ventanas similares provenientes de una misma canción y escribe en un archivo de salida las canciones detectadas.

Para implementar este comando debe buscar secuencias de ventanas de radio **Q** que sean similares a ventanas de una misma canción de **R** y que además mantienen un mismo desfase de tiempo. Por ejemplo, en la Figura 2 se puede ver que existe un segmento de tiempo de `radio_1.m4a` (entre los segundos 429 y 439) que es similar a la `cancion_03.m4a` (entre los segundos 6 y 16), porque los tiempos de las ventanas mantienen una diferencia constante de 423 segundos. Como cada emisión de una canción siempre ocurre en su totalidad, se puede concluir que `cancion_03.m4a` es emitida en `radio_1.m4a` a partir del segundo 423.

El comando debe reportar todas las apariciones de las canciones de **R** en los audios de **Q** generando un archivo de texto de **cuatro columnas**, separadas por un tabulador, con el siguiente formato:

Columna 1:	Nombre del archivo de radio de **Q**.
Columna 2:	Tiempo de inicio de la detección en el archivo de radio (en segundos).
Columna 3:	Nombre del archivo de canción de **R** que es audible en el archivo de radio a partir del segundo indicado.
Columna 4:	Valor de confianza de la detección. Es un número que mientras más alto, más seguro es que la detección sea correcta.

La siguiente figura muestra un posible archivo de detecciones generado por `tarea2-parte3.py`:

```
radio_1.m4a     87.3  cancion_08.m4a   2.29
radio_1.m4a    427.0  cancion_03.m4a   9.21
radio_1.m4a   1892.8  cancion_05.m4a   5.03
radio_1.m4a   2087.5  cancion_03.m4a   8.38
radio_2.m4a    192.8  cancion_07.m4a   5.03
radio_2.m4a   1235.1  cancion_03.m4a   9.10
```

Notar que es posible que una radio emita la misma canción más de una vez.

Datasets de prueba

Junto con este enunciado encontrará una implementación base para `tarea2-parte1.py`, `tarea2-parte2.py`, y `tarea2-parte3.py`, varios conjuntos de prueba (llamados `dataset_a`, `dataset_b`, `dataset_c`) que contienen audios de radioemisoras y trozos de canciones a buscar, y un programa de evaluación con la respuesta esperada para cada conjunto de consulta.

Evaluación

Antes de entregar su tarea, debe probarla con el programa de evaluación:

```
python evaluarTarea2.py
```

Este programa llama a `tarea2-parte1.py` dos veces (uno para los audios de canciones y otra vez para los audios de radio), luego llama a `tarea2-parte2.py` y finalmente llama a `tarea1-parte3.py`. Lee el archivo de texto generado por la parte 3 y compara las detecciones reportadas con el ground-truth del dataset, determina la distancia umbral de decisión que logra un mejor balance entre respuestas correctas y erróneas, y finalmente, dependiendo de la cantidad de detecciones correctas, señala la nota obtenida.

Una detección se considera correcta si intersecta una emisión del ground-truth que no ha sido detectada previamente.

Su tarea será evaluada en los datasets publicados y en otros datasets similares. Su tarea **no puede demorar más de 15 minutos** en procesar cada dataset de prueba.

Existe la posibilidad de obtener **hasta 1 punto extra de bonus** para otra nota si logra detectar correctamente la mayoría de las canciones en los datasets de evaluación.

Entrega

El plazo máximo de entrega es el **miércoles 04 de junio de 2025** hasta las 23:59 por U-Cursos. Existirá una segunda fecha de entrega sin descuentos, por definir.

La tarea la puede implementar en **Python 3** usando cualquier función de LibRosa, NumPy, SciPy y otras librerías gratuitas (exceptuando librerías de machine learning). Debe subir sólo código fuente de su tarea (archivos .py). No envíe datasets ni descriptores ya calculados.

Puede asumir que el comando ffmpeg está instalado.

Se recomienda incluir un archivo de texto señalando el sistema operativo en que realizó su tarea e incluir la salida que entregó `evaluarTarea2.py` en su computador.

Opcionalmente, puede implementar la tarea en C++ 17 usando cualquier función de la biblioteca estándar (std). Para calcular descriptores, se recomienda usar la clase Mfcc.hpp disponible en “Anexos Semana 05.zip” publicado en el Material Docente de U-Cursos. Debe subir el código fuente de su tarea junto con los pasos necesarios para la compilación.

La tarea es *individual* y debe ser de su autoría, es decir, no pueden ser resueltos por otro estudiante, no se pueden copiar respuestas de Internet, no se permite usar ChatGPT ni similares. En caso de detectar copia o plagio se asignará nota 1.0 a las o los estudiantes involucrados.