Tarea 2

CC5213 – Recuperación de Información Multimedia
Profesor: Juan Manuel Barrios
Fecha de entrega: 14 de octubre de 2024

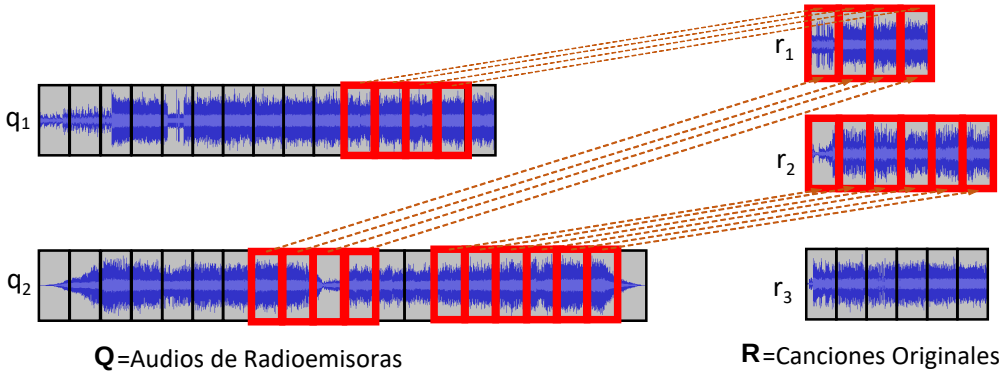
El objetivo de esta tarea es implementar un **detector de canciones emitidas por radioemisoras**. Dado un conjunto de canciones originales **R** y un conjunto **Q** con grabaciones de emisoras de radio, se desea determinar para cada archivo de audio de **Q** todas las apariciones de las canciones de **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 implementar su tarea deberá seguir los siguientes tres pasos. Primero, calcular descriptores por ventanas de todos los audios. Segundo, cada descriptor de audio de **Q** se compara con todos los descriptores de audio de **R** para determinar las ventanas más similares (k-NN). Tercero, procesar las listas de similares para localizar secuencias que provienen de una misma canción de **R**, determinar los segmentos de tiempo de **Q** en que aparecen canciones de **R,** y generar un archivo de emisiones con todas las canciones de **R** encontradas en **Q** con sus tiempos de aparición.

Comando 1: Extracción de características de audio

```
python tarea2-extractor.py [carpeta_audios_entrada]
                           [carpeta_descriptores_salida]
```

Este comando lee todos los archivos de audio (.m4a) que están en la carpeta de entrada, calcula descriptores de audio y escribe estos descriptores en uno o más archivos en la carpeta de salida.

Si lo necesita, utilice una herramienta como ffmpeg para convertir la pista de audio a otros formatos y use la carpeta de descriptores de salida para guardar archivos temporales.

Se recomienda dividir cada audio en ventanas de largo fijo y calcular un descriptor por ventana. Guarde en uno o más archivos los descriptores de todos los audios, incluyendo también datos de la ventana que representa cada descriptor. El formato, el contenido y la cantidad de archivos a guardar en la carpeta de salida es a su elección.

Comando 2: Búsqueda por similitud entre ventanas

```
python tarea2-busqueda.py [carpeta_descriptores_radio_Q]
                          [carpeta_descritores_canciones_R]
                          [archivo_ventanas_similares]
```

Lee los descriptores de los audios de radio **Q** y los descriptores de las canciones **R** (ambos generados por el comando anterior), calcula los vecinos más cercanos entre ellos y escribe un archivo conteniendo para cada ventana de radio **Q** la o las ventanas de canción más similares de **R**.

El formato y el contenido del archivo de salida es a su elección. Si lo desea, puede incluir datos adicionales, como por ejemplo, la distancia del más cercano o el segundo vecino más cercano.

La Figura 2 muestra el posible contenido del archivo de salida generado por el comando `tarea2-busqueda.py`. Notar que el formato del archivo a crear y el contenido específico es a su elección.

Figura 2: Posible archivo generado por `tarea2-busqueda.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_q_1.m4a	428.0	cancion_r_05.m4a	23.0
radio_q_1.m4a	**429.0**	**cancion_r_03.m4a**	**2.0**
radio_q_1.m4a	**430.0**	**cancion_r_03.m4a**	**3.0**
radio_q_1.m4a	**431.0**	**cancion_r_03.m4a**	**4.0**
radio_q_1.m4a	432.0	cancion_r_07.m4a	20.0
radio_q_1.m4a	433.0	cancion_r_08.m4a	3.0
radio_q_1.m4a	**434.0**	**cancion_r_03.m4a**	**7.0**
radio_q_1.m4a	435.0	cancion_r_07.m4a	20.0
radio_q_1.m4a	**436.0**	**cancion_r_03.m4a**	**9.0**
radio_q_1.m4a	**437.0**	**cancion_r_03.m4a**	**10.0**
radio_q_1.m4a	438.0	cancion_r_02.m4a	3.0
radio_q_1.m4a	**439.0**	**cancion_r_03.m4a**	**12.0**
radio_q_1.m4a	440.0	cancion_r_05.m4a	22.0

Comando 3: Detección de canciones

```
python tarea2-deteccion.py [archivo_ventanas_similares]
                           [archivo_detecciones]
```

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

El archivo de detecciones generado **debe tener formato de texto con cinco columnas** separadas por un tabulador (`\t`), cada detección en una línea, siguiendo el siguiente formato:

```
archivo_de_q \t tiempo_inicio \t largo \t archivo_de_r \t confianza
```

Los tiempos de inicio y largo se miden en segundos. El **valor de confianza** es un número que mientras más alto, más segura es que la detección sea correcta (por

ejemplo, podría ser el número de ventanas encontradas, el porcentaje de la canción encontrada u otro indicador que le parezca relevante).

La Figura 3 muestra un posible archivo de detecciones creados por este comando.

Figura 3: Ejemplo de archivo de detecciones creado por `tarea2-deteccion.py`. Cada línea tiene un nombre de archivo de **Q**, una ventana de tiempo (inicio y largo en segundos), el nombre de la canción de **R** audible en esa ventana y un valor de confianza.

```
radio_q_1.m4a      87.3    30.1    cancion_r_08.m4a   0.29
radio_q_1.m4a     427.0    14.0    cancion_r_03.m4a   7.21
radio_q_1.m4a    1892.8    25.9    cancion_r_05.m4a   5.03
radio_q_1.m4a    2087.5    30.1    cancion_r_03.m4a   2.38
radio_q_2.m4a     192.8    25.9    cancion_r_07.m4a   5.03
radio_q_2.m4a    1235.1    30.1    cancion_r_03.m4a   9.10
```

Para implementar este comando se recomienda buscar secuencias de ventanas de tiempo que provienen de una misma canción y que mantienen un mismo desfase de tiempo. Por ejemplo, en la Figura 2 se puede ver que existe un segmento de tiempo de `radio_q_1.m4a` (entre los segundos 429 y 439) donde es audible la `cancion_r_03.m4a` (entre los segundos 2 y 12) porque los tiempos de las ventanas similares mantienen una diferencia constante de 427 segundos.

Datasets de prueba y evaluación

Junto con este enunciado encontrará una implementación base para `tarea2-extractor.py`, `tarea2-busqueda.py`, y `tarea2-deteccion.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

El programa de evaluación se inicia con:

```
python evaluarTarea2.py
```

Este programa llama a `tarea2-extractor.py`, `tarea2-busqueda.py` y `tarea1-deteccion.py` sobre cada dataset de prueba, lee los archivos de resultados

generados, los compara con la respuesta esperada, 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.

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 todas las canciones en los datasets de evaluación.

Entrega

El plazo máximo de entrega es el **lunes 14 de octubre de 2024** 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 el Anexo 5.1 de “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. En caso de detectar copia o plagio se asignará nota 1.0 a los involucrados.