



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería
Informática**

SpotMyFM
spotmyfm.jorgeruizdev.com



Presentado por Jorge Ruiz Gómez
en Universidad de Burgos — 26 de junio
de 2022

Tutor: Bruno Baruque Zanón



UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática



D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Jorge Ruiz Gómez, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 26 de junio de 2022

Vº. Bº. del Tutor:

Vº. Bº. del co-tutor:

D. nombre tutor

D. nombre co-tutor

Resumen

Spotify es una de las plataformas de música en streaming más importantes del mundo. Se propone aumentar las funcionalidades de Spotify a partir de su API pública y otros servicios.

Se plantea el desarrollo y despliegue de una plataforma web que permita un usuario de Spotify explorar y gestionar su biblioteca de una manera más avanzada y transparente a la que ofrece Spotify por defecto.

La plataforma estará acompañada por una base de datos y un servicio de recuperación de información musical, que extraerá características de la señal de sonido de la canción y permitirá asociar géneros, subgéneros y estados de ánimo musicales a las distintas pistas; se utilizarán tres fuentes de datos, la API pública de Spotify, la API pública de LastFM y la API de SpotMyFM.

Descriptores

Recuperación de información musical, CNN, Spotify, plataforma web, sistema de recomendación.

Abstract

Globally, Spotify is one of the most important music streaming platforms. This project aims to improve Spotify's functionalities based on its public API and other services.

This project explores the development and deployment of a web platform that allows Spotify users to explore and manage their library in a more advanced and transparent way than the one offered by Spotify.

The platform will be supported by a database and a music information retrieval service (MIR), which will extract features from the sound signal of the song and associate genres, subgenres, and musical moods to the different tracks; three data sources will be used, Spotify's public API, LastFM's public API, and SpotMyFM API.

Keywords

Music information retrieval, CNN, Spotify, web platform, recommender system.

Índice general

Índice general	III
Índice de figuras	v
Índice de tablas	vi
Introducción	1
Objetivos del proyecto	5
2.1. Objetivos Generales	5
2.2. Objetivos Personales	5
2.3. Objetivos Técnicos	6
Conceptos teóricos	7
3.1. Clasificación mediante Redes Neuronales Convolucionales . .	7
3.2. Cuantización de Redes Neuronales	10
3.3. Ensembles, OVA y OVO	13
3.4. Ball Tree	15
3.5. Conceptos Sonido	15
Técnicas y herramientas	19
4.1. Tecnologías y Dependencias	19
4.2. Herramientas y Servicios	29
4.3. Técnicas y Metodologías	34
Aspectos relevantes del desarrollo del proyecto	35
5.1. Datasets	35
5.2. Unificación de las fuentes de datos	41

5.3. Aprendizaje Automático	45
5.4. Vecindad como Métrica de Similaridad de Audio	47
5.5. Arquitectura Contenedores	48
5.6. Pruebas e Integración Continua	50
5.7. Despliegue Continuo	50
Trabajos relacionados	53
6.1. Similitud de Audio Mediante Redes Neuronales Siamesas . .	53
6.2. Servicios Similares	53
Conclusiones y Líneas de trabajo futuras	57
7.1. Conclusiones	57
7.2. Líneas de Trabajo Futuras	59
Bibliografía	61

Índice de figuras

3.1.	Arquitectura de una red convolucional. Extraído de [1]	7
3.2.	Kernel 3x3. Extraído de [2]	8
3.3.	Aplicación del kernel 3x3 sobre un área 5x5. Extraído de [2] . . .	8
3.4.	Cuantización INT8 en el intervalo $[-0,5, 1,5]$. Extraído de [3] . .	11
3.5.	Resultados del coste de la operaciones dependiendo del tipo de dato de una red neuronal. Extraído de [3]	12
3.6.	Comparación entre OVA y OVO. Extraído de [4]	13
3.7.	Matriz para $N=4$	14
3.8.	MFCC de un fragmento de 30s de una canción. Extraído de spotmyfm/Ludwig/notebooks/gtzan/mfcc.ipynb	16
5.1.	Proceso de obtención de los datos de Ludwig Dataset	38
5.2.	Arquitectura del Proyecto	48
6.1.	Sistema de Recomendación basado en RNS. Extraído de [5] . . .	54

Índice de tablas

6.1. Comparativa entre Obscurify y SpotMyFM	55
---	----

Introducción

SpotMyFM es una plataforma diseñada para gestionar la biblioteca y playlists personales de un usuario de Spotify. La principal motivación para enfocar el proyecto alrededor de Spotify es que es una plataforma es muy conservadora a la hora gestionar playlists, bibliotecas y recomendaciones, ya que tienen como filosofía mantener al usuario en un entorno lo más familiar posible.

Este comportamiento se ve reflejado en servicios como Discovery Weekly¹, Mix o incluso el modo de reproducción aleatorio ya que tienden a favorecer canciones que has escuchado recientemente, evitando mostrar un gran número de canciones o artistas diferentes.

Esta filosofía ha sido objeto de estudio, como explora [6], llegando a la conclusión de que cuanto más familiar es el contenido para el usuario, más horas pasa en la plataforma.

Para compensar esta decisión de diseño y mejorar la experiencia de usuario de Spotify, se ha decidido diseñar e implementar una plataforma que permita gestionar la biblioteca personal del usuario, así como sus playlists, de una manera más avanzada.

La plataforma web incluye 7 servicios diferenciados:

1. **Gestión de Favoritos:** Es la página principal, muestra las canciones y artistas más escuchados en varios intervalos.
2. **Gestión de Biblioteca:** Permite explorar todas las características de la biblioteca personal del usuario²

¹Descubrimiento Semanal

²Las canciones marcadas como "Me Gusta"

3. **Gestión de Álbumes:** Permite gestionar y etiquetar los álbumes favoritos del usuario. El sistema de etiquetar álbumes se asemeja a la creación de listas de álbumes.
4. **Gestión de Playlists:** Permite explorar las playlists creadas o marcadas como me gusta por el usuario. Se pueden detallar cada una de las playlists.
5. **Análisis de Canción:** Permite subir y analizar una canción cualquiera. El analizador devuelve los géneros, subgéneros y estados de ánimo de la canción, así como 5 canciones similares.
6. **Búsqueda:** Permite buscar a cualquier artista, canción, álbum o playlist y disfrutar de las características que ofrece la plataforma.
7. **Reproductor:** Permite reproducir álbumes o canciones específicas, así como añadir a la cola, saltar o retroceder una canción.

Cada una de estos servicios incluye una vista con los distintos elementos en formato tarjeta interactiva o listado compacto, que pueden ser detallados. Esta lista puede ser filtrada (mediante texto) u ordenada por algún atributo.

1. **Canciones:** Esta vista permite filtrar canciones, crear o editar playlists y consultar las estadísticas generales y recibir recomendaciones para una selección de canciones.
Las canciones pueden ser detalladas con detalles como los géneros, estados de ánimo, fecha de lanzamiento, canciones similares, etc. Por otro lado incluye los detalles de los artistas y álbumes asociados, como las etiquetas de los álbumes, géneros de los artistas, descripción del álbum o un listado con las propias canciones que incluye el álbum de la canción.
2. **Playlists:** Esta vista permite listar todas las playlists del usuario. Cada playlist puede ser detallada junto a sus canciones, con todas las características de la vista anterior.
3. **Álbumes:** Esta vista permite filtrar álbumes, así como etiquetarlos. Los detalles de los álbumes son los mismos que los mencionados en el apartado de las canciones.
4. **Artistas:** Esta vista permite listar y detallar a los distintos artistas. Al detallar un artista se pueden explorar sus géneros, así como detallar los álbumes, singles y recopilatorios que ha publicado.

Para acompañar todas estas características se ha usado la API pública de LastFM, y se ha desarrollado Ludwig, un servicio que permite, a partir de un fragmento de una canción, obtener canciones similares en Spotify, géneros, subgéneros y los estado de ánimo de una canción. Este segundo servicio utiliza redes neuronales convolucionales (CNN) para clasificar las canciones a partir de su representación mediante sus coeficientes cepstrales en la frecuencia de mel.

Se ha desplegado el proyecto en spotmyfm.jorgeruizdev.com con una pequeña demo interactiva sin necesidad de registro con algunas de las funcionalidades.

Estructura de la Memoria

1. **Introducción:** Descripción del proyecto y estructura de la documentación.
2. **Objetivos del Proyecto:** Objetivos generales, personales y técnicos que se esperan cumplir con este proyecto.
3. **Conceptos Teóricos:** Explicación detallada sobre los conceptos más importantes del proyecto.
4. **Técnicas y Herramientas:** Listado y descripción de bibliotecas, servicios y técnicas utilizados durante el proyecto, así como una breve justificación y explicación de su uso.
5. **Aspectos Relevantes:** Listado de los puntos más interesantes e importantes que han surgido durante el desarrollo del proyecto.
6. **Trabajos Relacionados:** Exposición de trabajos y plataformas relacionadas con SpotMyFM.
7. **Conclusiones y Lineas de Trabajo Futuras:** Conclusiones sobre los distintos apartados del proyecto y un planteamiento de las mejoras que puede recibir el proyecto a lo largo del tiempo.

Estructura de los Anexos

- A. **Plan del proyecto:** Plan de proyecto software que estudia la planificación y viabilidad del proyecto.

- B. **Especificación de Requisitos:** Catálogo de los requisitos funcionales y no funcionales del proyecto.
- C. **Especificación de Diseño:** Diseño general de los distintos apartados del sistema así como su justificación.
- D. **Manual de Programador:** Manual con toda la información importante que permita a otro programador retomar el proyecto rápidamente.
- E. **Manual de Usuario:** Manual de usuario que explora todas las características de la plataforma web.

Otros Materiales

1. **Datasets:** Contiene los distintos conjuntos de datos utilizados para los sistemas de clasificación y recomendación.
2. **NextJS:** Contiene la aplicación de Frontend y Backend Principal.
3. **Ludwig:** Contiene las herramientas, notebooks y backend utilizados para el servicio de recuperación de la información musical y recomendación.
4. **Docker:** Contiene un fichero `compose.yml` que permite levantar todos los servicios automáticamente.

Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

2.1. Objetivos Generales

1. Diseñar e implementar un servicio web que extienda la capacidades de un servicio ya existente.
2. Diseñar e implementar una arquitectura que interconecte múltiples servicios
3. Diseñar e implementar un servicio que permita analizar y recomendar canciones a partir de un fichero de audio.

2.2. Objetivos Personales

1. Estudiar el diseño de una API comercial e integrarla en un proyecto.
2. Conocer el ciclo de desarrollo completo³ de un proyecto web.
3. Implementar, entrenar e integrar un modelo de deeplearning con una aplicación.
4. Generar un dataset robusto y variado.

³Análisis, especificación, diseño, desarrollo, pruebas, SEO y despliegue

5. Ampliar conocimientos de HTML5 y CSS3 a partir del diseño e implementación de componentes web.

2.3. Objetivos Técnicos

1. Desarrollar una aplicación web totalmente responsive⁴.
2. Desarrollar una aplicación web con modo claro y oscuro.
3. Poner en práctica los conocimientos de CI/CD ⁵
4. Crear un sistema de recomendación de música colaborativo y basado en contenidos integrado con Spotify.
5. Conocer frameworks y bibliotecas de pruebas, unitarias, automáticas y End to End (E2E).
6. Conocer los distintos conceptos y técnicas para trabajar con audio digital.

⁴Los contenidos se adaptan al dispositivo

⁵Integración Continua / Despliegue Continuo

Conceptos teóricos

3.1. Clasificación mediante Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una de las arquitecturas neuronales más utilizadas debido a su poder a la hora de trabajar con imágenes. Estas redes suelen estar formadas por una serie de bloques convolucionales, seguidos de una o varias capas densas en la salida de la red, como muestra 3.1

Bloque Convolutivo

Estos bloques, formados por capas, buscan resaltar los elementos más relevantes en los datos (normalmente imágenes) a partir de las operaciones de convolución y pooling.

La operación de convolución está formada por un kernel o filtro Fig.3.1 que se aplica Fig.3.3 a través de los datos. En el caso de la capa de convolución, los pesos entrenables de dicha capa son el propio kernel.

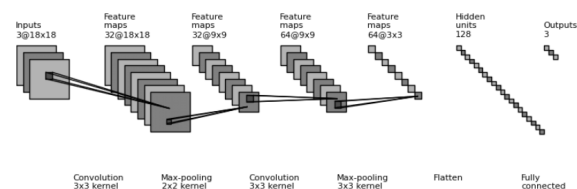


Figura 3.1: Arquitectura de una red convolutiva. Extraído de [1]

0	1	2
2	2	0
0	1	2

Figura 3.2: Kernel 3x3. Extraído de [2]

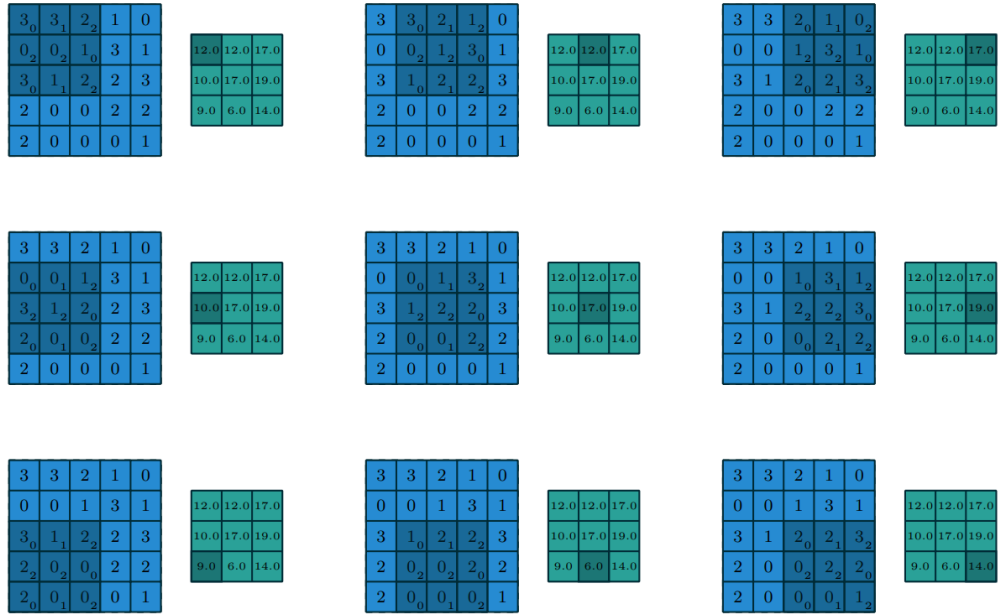


Figura 3.3: Aplicación del kernel 3x3 sobre un área 5x5. Extraído de [2]

Esta operación puede estar acompañada por una función de activación para cada una de las neuronas, como por el ejemplo la función de activación RELU (3.1).

$$Relu(x) = \max(0, x) \quad (3.1)$$

Por otro lado, la operación de pooling intenta reducir el tamaño de la entrada, dejando solo los datos relevantes de un área. Se aplica de la misma manera que una convolución, pero en vez de aplicar un kernel, realizamos

una operación aritmética sobre el área, como obtener el valor más grande del área (Max Pooling) o la media de todos los valores (Average Pooling)

Un bloque convolucional puede contener otras capas, como capas de normalización, que normalizan de nuevo los datos para trabajar siempre en el intervalo $[0,1]$ o $[-1, 1]$.

Densa

La salida de una red neuronal puede estar formada por capas densas, estas capas tienen una entrada unidimensional, por lo que si la salida del último bloque convolucional no es unidimensional, es necesario utilizar una operación de Flatten que transforma la matriz de entrada en una única fila concatenando cada una de las filas de la matriz.

En este tipo de arquitectura, cada una de las neuronas de una capa se interconectan con todas las neuronas de la capa siguiente.

Cada neurona está formada por:

- Entradas ($[x_1 \dots x_n]$)
- Salida (y)
- Vector de pesos entrenables ($[w_1 \dots w_n]$)
- Función de activación ($fn(x)$)
- Bias de activación entrenable ($[b_1 \dots b_n]$)

Y la salida (o salidas) de una neurona se calculan de la siguiente manera (3.2).

$$y = fn(x \cdot w + b) \quad (3.2)$$

La salida de cada una de estas neuronas se propaga a la siguiente capa como la entrada de cada neurona.

Capa de Salida

En tareas de clasificación, la capa de salida está formada por una o varias neuronas ⁶, y una función de activación.

⁶Depende del número de clases a clasificar

En tareas de clasificación con **varias clases**, utilizamos la función de activación Softmax, que aplica una normalización en la salida para resaltar la salida con la clase resultante.

Conociendo la salida de cada una de las neuronas, obtenemos la neurona con la salida más cercana a 1 (la salida más grande)

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, 2, \dots, n \quad (3.3)$$

Para obtener la clase, calculamos cual es la neurona con la activación más elevada.

$$clase = \operatorname{argmax}([y_0 \dots y_n]) \quad (3.4)$$

En tareas de clasificación con **varias etiquetas**, utilizamos la función de activación Sigmoide, que permite a cada neurona tener una salida entre 0 y 1, siendo este la confianza de que la clase contenga dicha etiqueta.

$$\operatorname{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (3.5)$$

Consideramos que una entrada contiene una etiqueta si el resultado de la sigmoide es mayor o igual que 0.5.

3.2. Cuantización de Redes Neuronales

La cuantización de redes neuronales es una de las técnicas de optimización de redes neuronales entrenadas más utilizada.

El proceso de cuantización consiste en convertir los distintos pesos (y por tanto operaciones) de la red neuronal a un tipo de datos con menos bits, lo que permite ahorrar memoria y acelerar las operaciones. Normalmente, la cuantización convierte los tensores de FLOAT32 (4 bytes por peso) a INT8 (1 byte por peso).

En el proceso de cuantización tenemos presente dos parámetros para cada uno de los tensores:

Escala (S): Es un número decimal que nos permite escalar los valores entre un número real y uno cuantizado.

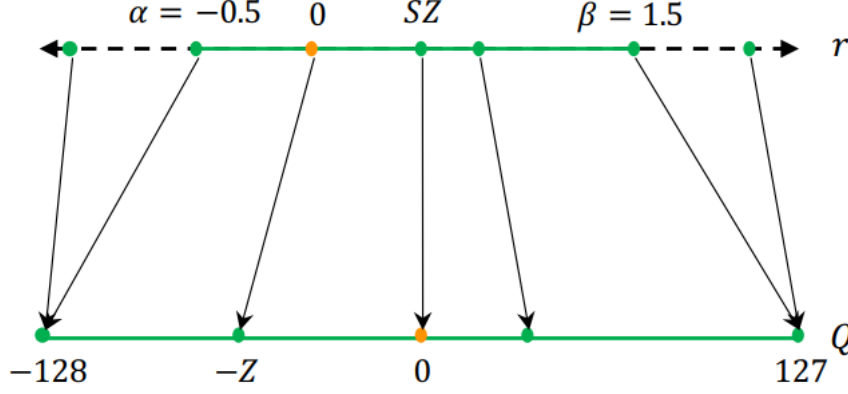


Figura 3.4: Cuantización INT8 en el intervalo $[-0.5, 1.5]$. Extraído de [3]

Zero Point (Z): Es un número dentro del intervalo de cuantización. Nos indica el valor del 0 real en valores cuantizados.

Estos valores son calculados a partir de analizar la distribución de los distintos tensores y activaciones.

$$S = \frac{\beta - \alpha}{\beta_q - \alpha_q} \quad (3.6)$$

$$Z = \text{round}\left(\frac{\beta\alpha_q - \alpha\beta_q}{\beta - \alpha}\right) \quad (3.7)$$

En el caso de la distribución de valores reales, debemos conocer el valor mínimo (α) y el máximo (β) del tensor a cuantizar.⁷

Para los tensores cuantizados, α_q y β_q se corresponden con el mínimo y el máximo de los tipos de datos, para INT8, $\alpha_q = -128$ y $\beta_q = 127$.

Una vez obtenidos S , Z , α_q y β_q , ya podemos cuantizar o decuantizar los distintos tensores con la siguiente ecuación.⁸

$$x_q = \text{clip}\left(\text{round}\left(\frac{1}{s}x + z\right), \alpha_q, \beta_q\right) \quad (3.8)$$

⁷Existen otras técnicas más avanzadas que tienen en cuenta la existencia de outliers en la distribución, lo que puede aumentar el ruido de cuantización.

⁸La operación de decuantización debe tener en cuenta los errores de redondeo y clip si se quiere recuperar x . Nosotros no tenemos en cuenta los errores ya que no afectan al rendimiento de la red.

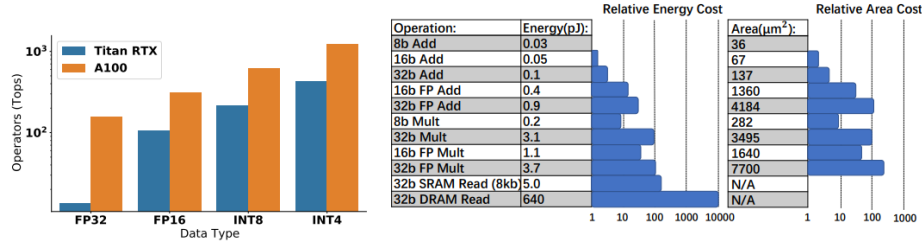


Figura 3.5: Resultados del coste de la operaciones dependiendo del tipo de dato de una red neuronal. Extraído de [3]

Siendo clip una operación de saturación que impide que resultado de la cuantización sea un valor sea menor que α_q o mayor que β_q . En la figura 3.4 esta operación se ve reflejada en los valores que están fuera del intervalo $[\alpha, \beta]$

Técnicas de Cuantización

Existen dos técnicas de cuantización post entrenamiento:

Fake Quantization o Cuantización Dinámica:

Los pesos se almacenan en un tipo de datos inferior, pero los parámetros de cuantización S y Z se calculan en tiempo de inferencia.

Fixed Quantization:

En este caso los parámetros de la cuantización se calculan durante el proceso de cuantización. Este tipo de cuantización requiere utilizar un **conjunto de datos representativo de entrada** durante el proceso de cuantizado, con el que se calcularán los parámetros de cuantización de cada tensor.

Resultados de la cuantización

Si bien este proceso es muy eficiente como se puede observar en la figura 3.5, tiene un inconveniente muy importante, y es que la salida de una neurona está limitada a tantos valores como como el tipo de dato que utilizamos. En el caso de INT8 o UINT8, una neurona solo puede tener 256 valores distintos en su salida.

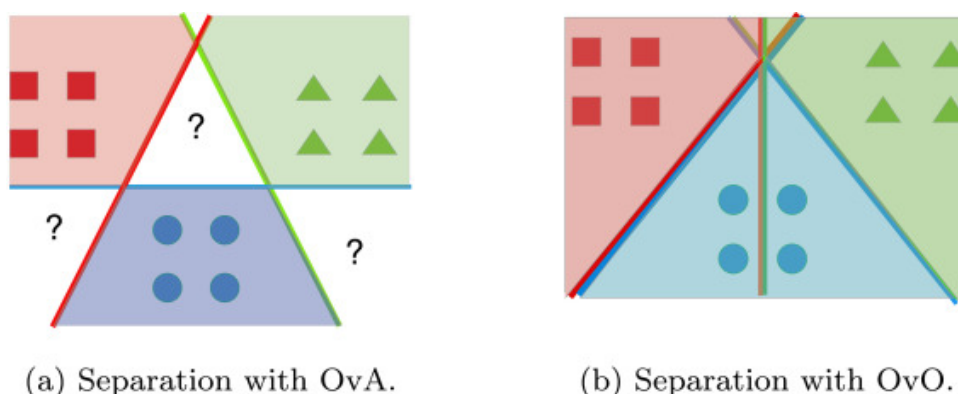


Figura 3.6: Comparación entre OVA y OVO. Extraído de [4]

Por otro lado, esto no debería ser ningún problema si trabajamos con tareas de clasificación con pocas etiquetas, pero es importante comprobar con un conjunto de prueba que la precisión de la red no se ha visto afectada por la cuantización.

3.3. Ensembles, OVA y OVO

Existen múltiples técnicas para clasificar conjuntos de datos con múltiples clases. En este apartado vamos a comentar dos tipos de ensembles que unen varios clasificadores binarios para obtener un único clasificador multiclase. En algunas situaciones, los ensembles binarios pueden ser mucho más robustos que un único clasificador multietiqueta.

One vs All

El ensemble One vs All (OVA) divide un problema con N clases en N clasificadores binarios. Cada uno de estos clasificadores se encarga de identificar si una entrada se corresponde con una clase (1) o no (0). En este caso, es importante que el clasificador no esté calibrado, es decir, su salida debe asemejarse con la confianza del resultado, ya que, si nos encontramos con un conflicto entre varios clasificadores base por tener la misma salida, no vamos a poder predecir la clase correctamente.

Ejemplo Red Neuronal Binaria

Si utilizamos la **función de activación escalón** en la neurona de salida de nuestra red neuronal, nos encontramos con que la salida únicamente tiene dos valores, 0 y 1, por lo que no conocemos la confianza de la predicción.

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

Figura 3.7: Matriz para N=4

Por otro lado, si utilizamos la **función sigmoide** (3.5) como función de activación, nos encontramos con una función cuyo recorrido son valores reales en el intervalo $[0, 1]$, por lo que podríamos considerar que la salida es la confianza que tiene la red ante una predicción.

One vs One

El ensemble One vs One (OVO), a diferencia del ensemble OVA, busca una clasificación más robusta entrenando un mayor número de clasificadores. Si en el ensemble OVA únicamente lidiábamos con las colisiones teniendo en cuenta la confianza, el ensemble OVO es capaz de conocer si un dato pertenece a una clase o a otra, ya que cada uno de los clasificadores se entrena con un subconjunto de datos formado solo por dos clases⁹.

En este caso, tenemos que entrenar $L = K(K - 1)/2$ clasificadores binarios, un número mucho más grande que en el ensemble OVA, por lo que el coste en memoria y tiempo de entrenamiento es mucho mayor.

En el caso del ensemble OVO, el clasificador base no solo no tiene que estar correctamente calibrado, sino que además las salidas deben estar comprendidas en el intervalo $[-1, 1]$, siendo -1 la clase A, 1 la clase B, y 0 el valor correspondiente la clase desconocida.

Para evaluar la salida de todos los clasificadores, debemos construir una matriz $N \times L$ (3.3), que permita convertir el vector L a un vector con N elementos con la confianza de cada clase. Cada fila de la matriz se corresponde con cada clase, y debemos asignar el mismo valor (1 o -1) que hemos utilizado como etiqueta durante el entrenamiento.

Haremos una multiplicación de matrices entre la matriz y la salida de los clasificadores, obteniendo una matriz $N \times L$ con la probabilidad de cada clase. Como hemos utilizado valores en el intervalo $[-1, 1]$ como salida de

⁹Clasificación Binaria

cada clasificador, al realizar la multiplicación de matrices nos encontraremos con que únicamente tenemos valores positivos.

Si sumamos todas las columnas, de la matriz anterior obtendremos la confianza del ensemble para cada uno de los datos.

3.4. Ball Tree

Un ball tree o árbol de bolas es una estructura de datos jerárquica utilizada para la gestión de vecindarios de instancias. Esta estructura de datos permite conocer de una manera eficiente el vecindario de una instancia. Durante la construcción del árbol, las instancias se dividen en dos clusters o bolas¹⁰, siendo cada cluster una rama del árbol. Los clusters son calculados de la siguiente manera:

1. Se selecciona una instancia aleatoria
2. Se busca la instancia más alejada a la instancia seleccionada
3. Se calcula el punto intermedio que hay entre las dos instancias
4. Se divide el espacio en dos mediante un hiperplano
5. Se calculan los centroides de los clusters que hay en cada división.
6. Se crea una esfera que cubra todas las instancias de cada clúster

Repetimos el proceso para cada una de las esferas hasta alcanzar el número de instancias por hoja buscado (Por defecto son 40 instancias por hoja en SKLearn).

Hecho esto, podemos consultar rápidamente el vecindario correspondiente a una nueva instancia a partir de los centroides más cercanos a nuestra instancia, sin necesidad de explorar todos los vecinos posibles.

3.5. Conceptos Sonido

Onda Digital y Frecuencia de Muestreo

Una onda analógica es una señal continua a lo largo del tiempo. Para poder trabajar con una onda digitalmente, es necesario discretizar la señal

¹⁰También conocido como esferas o hiperesferas

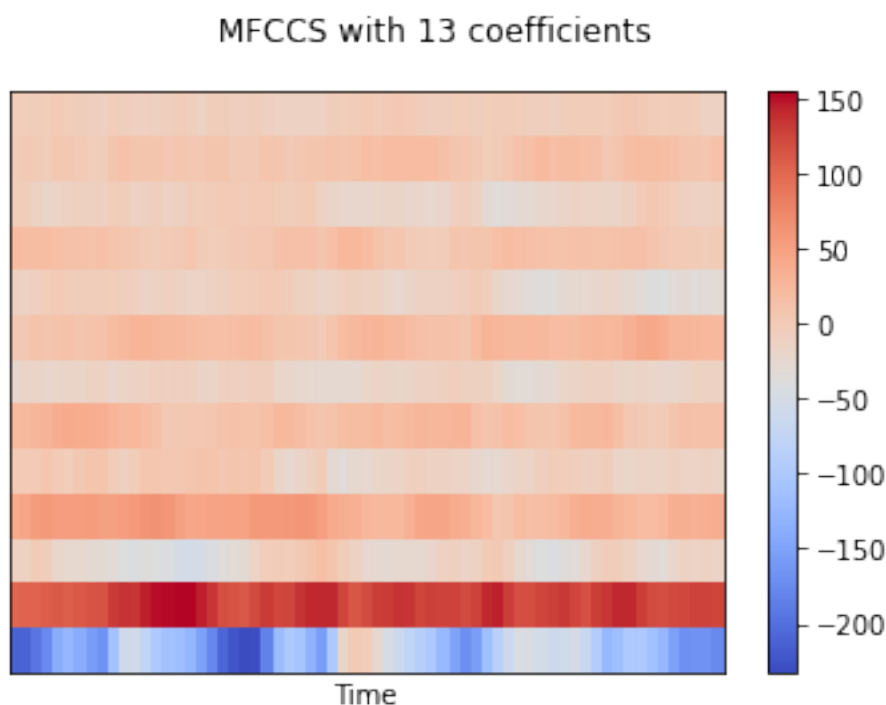


Figura 3.8: MFCC de un fragmento de 30s de una canción. Extraído de [spotmyfm/Ludwig/notebooks/gtzan/mfcc.ipynb](https://spotmyfm.com/Ludwig/notebooks/gtzan/mfcc.ipynb)

analógica en una señal digital. Para ello, se declara una frecuencia de muestreo que indica cada cuantas medidas discretas se realizan sobre la amplitud de la onda en una señal analógica a lo largo del tiempo. En nuestro caso vamos a trabajar con una frecuencia de muestreo de 22050Hz , es decir, se van a tomar 22050 puntos discretos de la amplitud de una onda cada segundo.

Este proceso de discretización de una señal digital es conocido como **modulación por impulsos codificados**. El formato de audio .wav almacena el audio sin comprimir directamente con esta modulación, por eso será utilizado a lo largo del proyecto en vez de otros formatos más eficientes como .mp3.

Coeficientes Cepstrales en las Frecuencias de Mel

Los coeficientes cepstrales en las frecuencias de mel o MFCC Fig.3.8 son un tipo de representación de la densidad espectral del sonido que intentan representar la percepción auditiva humana.

Son muy utilizados en tareas de reconocimiento del habla y recuperación de información musical.

Estos coeficientes están ajustados sobre la **escala de mel**, una escala que aproxima la escala de tono a una escala similar a la percepción humana, mejorando la representación del sonido.

Para obtener los MFCC partimos del espectrograma de mel, un espectrograma ajustado a la escala de mel. En este caso, estamos intentando obtener el espectrograma de un espectrograma, también conocido como cepstrum.

Un método para calcular los MFCCs puede ser el siguiente:

1. Separar la señal en segmentos (hop_length)
2. Aplicar la Transformada Discreta de Fourier a cada segmento para obtener la **potencia espectral**.
3. Transformar los resultados del paso 2 a la escala de mel.
4. Aplicar el logaritmo a cada frecuencia de mel, (escala logarítmica en **decibelios**).
5. Aplicar la Transformada Discreta del Coseno al resultado del paso 4
6. Escoger N MFCCs¹¹.

Los pasos 1-3 se corresponden con la obtención de un espectrograma de mel.

Se recomienda consultar el apéndice A de [7] para consultar los detalles de la extracción.

¹¹En 3.8 se han escogido $N = 13$ MFCCs

Técnicas y herramientas

4.1. Tecnologías y Dependencias

Frontend

Typescript

Typescript es un superset del lenguaje de programación Javascript, por lo que cualquier código escrito en Javascript puede ser utilizado como código Typescript. La principal característica de este superset es la inclusión de un sistema de tipado estático fuerte.

Para el desarrollo de los componentes se ha utilizado TSX (Typescript Syntax Extensions), esta extensión de Typescript permite generar HTML junto con Typescript, facilitando la generación de HTML dinámico. Estas extensiones de lenguaje no están reflejadas en el estándar de ECMAScript, por lo que es necesario utilizar un transpilador que convierta estas extensiones en código Javascript.

Se ha tenido en cuenta utilizar JavaScript y JSX como alternativa.

NextJS

NextJS es un framework de Javascript compatible con Typescript que permite construir interfaces. NextJS está construido sobre la biblioteca ReactJS, y añade una gran variedad de características como:

- Rutas o Páginas (/home, /about, etc)
- Server Side Rendering
- Rutas de API basadas en NodeJS

- Empaquetamiento sin configuración
- Api de internacionalización

ReactJS es una biblioteca de desarrollo de interfaces web que incluye un Virtual DOM que permite montar y desmontar componentes JSX en el DOM de manera transparente.

Se ha tenido en cuenta utilizar ReactJS como una alternativa.

Tailwind CSS

Tailwind CSS es una biblioteca de generación de hojas de estilo CSS de código abierto muy interesante. Su principal filosofía es la de la creación de clases de utilidad dinámicas con las que podemos seguir buenas prácticas de CSS. Esta biblioteca está acompañada con un preprocesador que se encarga de generar y optimizar las stylesheets a partir de las clases que hemos utilizado.

Una de sus características más interesantes es la generación de clases de utilizado con variantes que permiten cambiar el estilo dependiendo de algunos condicionales

Algunas de estas variantes de estado pueden ser:

- `hover:<clases de utilidad>` estilo cuando el ratón está encima de un elemento
- `sm:<clases de utilidad>` estilo cuando el ancho del viewport es mayor a 640px
- `dark:<clases de utilidad>` estilo a aplicar cuando estamos utilizando el modo oscuro
- ...

Por último, esta biblioteca permite declarar nuestras propias paletas de colores, variantes de estado, clases de utilidad, etc, por lo que podemos expandirla manualmente.

Twin Macro y Styled Components

En la actualidad, debido a las limitaciones de las hojas de estilo en cascada, existen múltiples métodos de escribir estilos CSS

- Ficheros .css
- Estilos en línea
- Estilos como componentes
- CSS in JS

Para esta aplicación hemos utilizado el paradigma CSS in JS, que nos permite declarar los estilos CSS en el propio código de la aplicación. La principal ventaja de este paradigma es que podemos modificar los atributos CSS directamente desde nuestro código.

Twin macro es un adaptador de distintas bibliotecas **CSS in JS**, que permite estilar mediante TailwindCSS con bibliotecas CSS in JS como PostCSS, Emotion o Styled Componets.

En este caso hemos escogido la biblioteca **Styled Coponentes**, que nos permite declarar o estilar componentes de ReactJS.

Zustand

Zustand es una biblioteca de código abierto para la gestión de estado de React basada en hooks. Esta biblioteca permite declarar stores (espacios de almacenaje de estados) en los que podemos almacenar datos y sus operaciones que deseamos compartir entre varios componentes.

En el caso de la aplicación, hemos utilizado esta biblioteca para almacenar estados globales, como el estilo de la web, la pestaña que tenemos activa o instancias únicas de los distintos clientes REST de la aplicación.

DexieDB

DexieDB o DexieJS es un wrapper de IndexedDB, una **base de datos NO-SQL** que se encuentra en todos los navegadores modernos.

Este wrapper, a parte de contener toda la funcionalidad de DexieDB, nos permite declarar los esquemas de la base de datos junto sus distintas operaciones y comprobaciones.

DexieDB es compatible con Typescript, por lo que permite operar las tablas con supervisión de tipos.

Jest

Jest es una biblioteca de código abierto de pruebas unitarias desarrollada por Facebook. Esta biblioteca tiene una gran variedad de características, como Mocking, compatibilidad con código asíncrono, compatibilidad con React, Typescript, Node, etc.

Una de las principales ventajas de Jest son las extensiones de su comunidad. En este caso se ha utilizado la extensión **React Testing Library**, una extensión que permite renderizar internamente componentes de React y hacer comprobaciones sobre el DOM Virtual.

Cypress

Cypress es un framework de pruebas punto a punto de código abierto. Cypress es muy similar a Selenium, ambas realizan pruebas automatizadas sobre un navegador físico, pero Cypress es mucho más sencillo de configurar.

Algunas de las características más relevantes de Cypress son:

- Mayor velocidad de ejecución
- Tests escritos en Javascript
- Dashboard y grabación de ejecución de pruebas
- Tamaño de viewport variable (Tests para tablets, móviles, escritorio, etc)
- Ejecución headless
- Compatibilidad con Github Actions

Otras:

- **Recharts**: Es una biblioteca de código abierto para React que permite generar gráficas interactivas.
- **React Icons**: Una colección de iconos vectoriales que junta múltiples bibliotecas de iconos.
- **Framer Motion**: Biblioteca de código abierto que permite añadir animaciones que dependen de los estados internos de un componente.

Backend NextJS

NextJS API REST

NextJS incluye una API REST funcional basada muy simple basada en Node/Express. Esta API tiene como principal objetivo apoyar al funcionamiento del cliente en dos situaciones.

- Renderizar en el servidor (SSR)¹² elementos dinámicos de la web. Como por el ejemplo vistas individuales para usuarios registrados.
- Trabajar con peticiones que requieran autenticación.

En este api, un endpoint se define como un fichero .ts/.js que recibe dos objetos, la petición HTTP y la respuesta HTTP. Como cada endpoint se corresponde con un fichero único, la API permite el paradigma de cloud computing FaaS (Functions as a Service), en el que cada función puede actuar como un pequeño microservicio.

Cada endpoint puede hostearse de manera independiente en plataformas como Cloudflare Workers o AWS Lambda.

JWT

JWT (Json Web Tokens) es un método de representación de tokens en formato json basado en el estándar RFC 7519. Los tokens están formados por tres segmentos:

- Cabecera: Es un json serializado en base64 que incluye el tipo del token y el algoritmo de firma utilizado (RSA, SHA256, etc)
- Payload: Es otro json serializado en base 64 que incluye los contenidos del token.
- Firma de verificación: Firma hash generada a partir de una clave privada y los campos anteriores. Permite comprobar que el payload no ha sido modificado.

Una vez contruidos y serializados los 3 segmentos, estos son concatenados mediante un punto (".") y pueden ser deserializados por cualquier receptor.

¹²Server Side Rendering

DynamoDB y Dynamoose

DynamoDB es una base de datos NO-SQL basada en documentos clave valor desarrollada por Amazon para AWS, su plataforma de Cloud Computing.

Algunas características de esta base de datos son:

- Capacidad bajo demanda
- Base de datos sin servidor (No utiliza sockets / conexiones físicas, por lo que es apta para ser accedida por la arquitectura sin servidor, como CaaS o FaaS).
- Compatibilidad con AWS OpenSearch (ElasticSearch)
- Copias de seguridad
- Replicas
- Transacciones ACID

Además, DynamoDB está incluida en el plan gratuito de AWS, y ofrece 25GB de por vida.

Dynamoose es una herramienta de modelado muy similar a Mongoose, que permite utilizar las distintas características de AWS desde un api de alto nivel.

La principal ventaja de Dynamoose es que permite modelar los esquemas de la base de datos en Typescript/Javascript. Estos esquemas serán validados por Dynamoose antes de crear, modificar o leer documentos.

Otras Bases de datos planteadas

	Firestore	MongoDB	Supabase	DynamoDB	Heroku
Tipo	Documentos	Documentos	Relacional	Documentos	Relacional
Capacidad	1GB	0.5GB	0.5GB	25GB	1000 filas
Lim. Peticiones	50k R/doc/day 20k W/ doc/day	No hay	No Hay	200M R+W /mes	No
Self-hosted	No	Si	Si	NO	Si
SSL	Si	Si	Si	Si	Si
Escalable	Si	Si	Si	Si	Si
Pausa	No	No	Si (7 días)	No	Si (7 días)
ORM/ Bussiness Logic / Model / Validation	No	Si (Mongoose)	No	Si (Dynamoose)	Si (Postgres)
Editor	Si	Si	Si	Si (De pago)	Si

Backend MIR

El backend de **Music Information Retrieval (MIR)** está basando en Python 3.8

FASTAPI

FastAPI es un framework web diseñado para construir APIs con Python 3.6+.

La principal característica de FastAPI es que utiliza el sistema de sugerencia de tipos de Python para validar las distintas peticiones HTTP.

Este sistema permite asignar un tipo de forma estática a una variable, estos tipos son ignorados en tiempo de ejecución, pero pueden ser utilizados para documentar o como comprobación en tiempo de desarrollo mediante un linter.

```
foo: int = 1 # No error
foo: int = "bar" # typechecking error
```

Además de validar las peticiones de nuestra aplicación, FastAPI genera automáticamente **documentación interactiva** para cada endpoint con **OpenAPI** y **Redocs**.

Por último, FastAPI utiliza el api de **corrutinas** de Python basada en la sintaxis Async/Await. Cada uno de los endpoints puede ser definido como una corrutina asíncrona con la que podemos esperar a ejecución de otras corrutinas utilizando la palabra clave await, facilitando el uso de bibliotecas concurrentes.

AIOHTTP

aiohttp es una biblioteca de Python basada en **requests** que permite realizar **peticiones http concurrentes**. aiohttp utiliza el api de corrutinas, por lo que puede ser consumida directamente desde un endpoint de FastAPI.

ONNX Runtime Python

ONNX Runtime [8] es una biblioteca de código abierto mantenida por Facebook y Microsoft. Esta biblioteca implementa múltiples motores de inferencias compatibles con el estándar **Open Neural Network eXchange**

(**ONNX**), un estándar que busca la interoperabilidad de modelos neuronales a partir de una **representación intermedia (IR)**.

La filosofía de ONNX es que un modelo neuronal pueda ser utilizado en cualquier hardware independientemente de su representación interna, para ello existen una serie herramientas para convertir y optimizar modelos basados en Pytorch, Keras, Caffé, etc en representación intermedia.

Un modelo IR puede ser utilizado en múltiples **plataformas**

- Windows
- Linux
- Android
- IOS
- Navegador
- ...

Independientemente de la **arquitectura**

- X86
- X64
- ARM64
- ...

Desde múltiples **lenguajes de programación** o **entornos de ejecución**

- Python
- NodeJS / Javascript / Typescript
- C++
- Java
- ...

Y compatible con una gran variedad de APIs para la aceleración por Hardware

- CPU

- Nvidia Cuda
- Nvidia TensorRT
- Intel OpenVINO
- Windows DirectML
- ...

Librosa

Librosa es una biblioteca de Python enfocada al análisis de audio y música. Esta biblioteca está basada en Scipy y es muy utilizada a la hora de trabajar con sonido en Python.

Esta biblioteca incluye todos los componentes necesarios para poder trabajar con audio, desde la lectura / creación de ficheros de audio, conversión entre formatos, muestreo, visualización y extracción de características.

En este apartado del proyecto utilizamos librosa para leer ficheros wav y extraer algunos datos como:

- MFCCs
- Tempo
- Beat
- Descomposición del espectrograma

Se tuvo en cuenta el uso de Torch Audio para las tareas de preprocesamiento.

FFMPEG

FFMPEG es conjunto de bibliotecas de **alto rendimiento** que permiten trabajar con datos multimedia (ficheros y streams de vídeo, audio, imágenes, etc). FFMPEG permite grabar, editar, convertir, transcodificar o escalar datos multimedia.

Ludwig utiliza FFMPEG para convertir y muestrear ficheros .mp3 a .wav en 22050HZ, ya que es la representación de onda en formato .wav es la misma con la que trabaja Librosa. Librosa es compatible con formatos .mp3, pero su implementación no es tan eficiente como la de FFMPEG.

Sklearn

Sklearn [9] es una biblioteca basada en SciPy [10] diseñada para tareas de aprendizaje automático.

Incluye una gran variedad de algoritmos para tareas de clasificación, regresión, clustering, reducción de dimensionalidad, selección de modelo o preprocesado.

Este backend utiliza dos componentes de Sklearn:

- **BallTree**: Utilizado para encontrar los vecinos más cercanos a un dato.
- **MinMaxScaler**: Utilizado para normalizar la entrada al BallTree

Knime

Knime o Konstanz Information Miner es una herramienta de código abierto usada en minería de datos y aprendizaje automático. Esta herramienta permite construir de manera visual mediante nodos interconectados flujos de trabajo para visualizar, modificar y analizar datos. Se ha utilizado KNime para el análisis de los distintos conjuntos de datos.

Scikit Surprise

Surprise [11] es una biblioteca de código abierto especializada en sistemas de recomendación colaborativos.

El funcionamiento de esta biblioteca es muy simple, ya que permite generar la matriz de recomendación a partir de una lista formada por:

- Id Usuario
- Id Elemento
- Votación numérica del Usuario

Surprise implementa una gran variedad de algoritmos, como SVD o NMF utilizados en descomposición de matrices, y otros más tradicionales como KNN, Co Clustering o Slope One.

Por otro lado, Surprise además implementa un sistema de evaluación de recomendaciones, que valora la precisión del modelo a partir de intentar completar recomendaciones para usuarios incompletos.

4.2. Herramientas y Servicios

Kaggle

Kaggle es una comunidad de ciencia de datos en la que se publican conjuntos de datos y competiciones.

Uno de los servicios que incluye Kaggle es **Code**, un servicio para publicar y ejecutar notebooks que utilicen el Dataset, esta ejecución se realiza en unos contenedores virtuales con las siguientes características:

- 4 Núcleos
- 12GB de RAM
- NVIDIA TESLA P100 de 16GB (opcional)
- Google TPU (opcional)

En este caso se ha utilizado Kaggle para publicar los datasets y entrenar los modelos de forma remota.

Se ha planteado el uso de Google Cloud Notebooks y Google Colab para el entrenamiento de los modelos.

Tensorflow Keras

Tensorflow [12] es una biblioteca de código abierto destinada a el entrenamiento e inferencia de modelos neuronales. Tensorflow está pensada para ser usada con Python, pero es compatible con otros lenguajes como C++ o incluso JavaScript.

Keras es una API de Tensorflow para Python que simplifica la implementación y entrenamiento de redes neuronales mediante Tensorflow.

Se ha planteado el uso de Pytorch como alternativa a Tensorflow Keras.

ONNX Tensorflow Converter y ONNX Quantizer

ONNX Tensorflow Converter [8] es una herramienta de línea de comandos que permite convertir un modelo de Tensorflow en un modelo del estándar ONNX.

ONNX Quantizer es una biblioteca de Python que permite cuantizar modelos de ONNX. Esta herramienta es compatible tanto con la cuantización dinámica como con la estática.

Ambas herramientas forman parte del proyecto de código abierto ONNX, mantenido por Microsoft y Facebook.

Se ha tenido en cuenta el uso de TFLITE para cuantizar e inferir los modelos, pero este motor únicamente está optimizado para dispositivos móviles (ARM).

Github

Github es una plataforma para hostear repositorios git. Esta plataforma permite crear y compartir repositorios públicos y privados, utilizar ramas, crear y gestionar pull requests, y muchas otras características como la creación de flujos de trabajo con Github Actions o el control de dependencias con Dependabot.

Se ha utilizado Github y su sistema de Issues para el control de versiones del proyecto y la planificación temporal del proyecto.

Zenhub

Zenhub es una herramienta de gestión de proyectos que intenta integrarse con Github. Esta biblioteca presenta un tablero Kanban en el que podemos encontrarnos cada una de las tareas del propio repositorio.

Zenhub, además tiene un sistema de sprints, épicas y puntos de historia con el que podemos gestionar proyectos utilizando la metodología Scrum.

Dependabot

Dependabot es un servicio integrado en Github que permite analizar un repositorio en busca de brechas de seguridad.

Este servicio analiza el código en busca de **filtraciones de claves o tokens secretos**, y como su propio nombre indica, analiza las **dependencias del proyecto** en busca de dependencias vulnerables.

Si Dependabot detecta una vulnerabilidad en el proyecto, este intentará corregirla y publicará una **Pull Request** con la vulnerabilidad solucionada.

Docker

Docker es una plataforma que permite crear y desplegar contenedores. Un contenedor es un “paquete” que contiene todos los requisitos para que una aplicación se pueda ejecutar en un entorno aislado, como el binario de la aplicación, dependencias, configuraciones de red, etc.

Estos contenedores pueden ser considerados una pequeña máquina virtual, ya que cada contenedor tiene su propio sistema de ficheros, unidades, o LAN, pero a diferencia de una máquina virtual, el contenedor se ejecuta sobre el **Docker Engine** y no sobre un **hipervisor**.

En este caso se ha utilizado Docker para crear los contenedores con los que poder desplegar las aplicaciones de NextJS y FastAPI.

Github Copilot

Github Copilot es un servicio online que permite sugerir cambios o implementaciones de código basado en el contexto actual del fichero que estamos editando. Copilot se integra con el editor (VSCODE) o IDE (Webstorm) y necesita conexión a internet para procesar las sugerencias.

La principal ventaja de Copilot frente a otros servicios de autocompletado como Intellisense es que Copilot utiliza los tipos de datos, comentarios, estilo del código, nombres de variables y nombres de funciones para sugerir cambios o implementaciones.

Github Actions

Github Actions es un servicio integrado en Github que permite ejecutar flujos de trabajo sobre un repositorio a partir de distintos estados.

En este caso, se han programado las siguientes acciones para cada push al repositorio:

- **Prettier:** Aplica el formateador de código Prettier a todos los ficheros compatibles (Typescript, Javascript, JSON, CSS, etc)
- **Integración Continúa:** Asegura que el proyecto se compila correctamente y ejecuta las pruebas unitarias sobre las distintas clases del proyecto.
- **Cypress:** Ejecuta las pruebas punto a punto sobre el despliegue de la aplicación.

Se ha considerado el uso de los servicios TravisCI y CircleCI.

Vercel (Plataforma)

Vercel es una plataforma para publicar proyectos de frontend y webs estáticas. Está desarrollado por la empresa Vercel, la misma empresa que desarrolla NextJS 4.1, por lo que es compatible con todas las funcionalidades de NextJS.

Vercel ofrece muchas otras características como:

- Compatibilidad con **Edge Functions** (FaaS)
- Despliegue Continuo desde un repositorio
- SSL
- Dominios Personalizados

Se ha planteado el uso de la plataforma Netlify pero se ha escogido Vercel por su integración con NextJS.

Google Cloud Run

Google Cloud Run es un servicio de Google Cloud que permite el despliegue de contenedores (CaaS)¹³ de forma escalable. Su funcionamiento es muy simple, a partir de un contenedor con un puerto expuesto, Cloud Run creará un proxy para dicho puerto y dependiendo del tráfico el orquestador instanciará o mantendrá activos un número variable de contenedores.

Visual Studio Code

Visual Studio Code es un editor de código de código abierto desarrollado por Microsoft. Este editor está construido sobre Chromium y Electron, con una filosofía en la que la interfaz está totalmente separada del editor, por lo que el editor en si puede encontrarse en un servidor, máquina virtual o local, mientras que editamos desde un navegador web o la propia aplicación de escritorio.

La principal ventaja de Visual Studio Code es su gran comunidad, que ha desarrollado un gran número de extensiones para ajustar el desarrollo del editor para cada uno de los casos de uso.

¹³Containers as a Service / Contenedores como Servicio

OpenAPI

La especificación OpenAPI es una especificación para definir la interfaz de un servicio REST. Esta interfaz puede declararse mediante un fichero .json o .yaml, y permite declarar los distintos endpoints de un servidor, así como las peticiones y respuestas esperados.

La principal ventaja de OpenAPI es que puede ser integrada con un middleware para ofrecer validación de la API.

Por otro lado, la especificación OpenAPI permite generar una interfaz interactiva con la que poder generar automáticamente peticiones de muestra a partir de la especificación de peticiones.

Lucidchart

Lucidchart es una aplicación web progresiva diseñada para la creación y exportación de diagramas. Su principal característica es su gran cantidad de herramientas, así como la posibilidad de editar un diagrama de forma colaborativa. Lucidchart dispone de un plan gratuito que incluye hasta tres proyectos, cada uno puede albergar infinitos diagramas.

Se ha planteado el uso de draw.io, pero se ha preferido usar Lucidchart por su servicio en la nube.

Overleaf y L^AT_EX

Overleaf es un editor web de documentos L^AT_EX. L^AT_EX es una extensión de T_EX, un sistema de composición de textos orientado al ámbito científico y matemático, es ampliamente utilizado debido a su estilo y a su procesador de expresiones matemáticas. L^AT_EX, a diferencia de otros sistemas como DOCX, está pensado para que los textos sean editados directamente en texto plano, sin necesidad de un editor específico. Esto es posible gracias al sistema de macros y comandos que permiten estilar y organizar un documento.

Figma

Figma es una aplicación web progresiva destinada a la edición de imágenes vectoriales. Su caso de uso principal es de prototipado de interfaces de usuario, pero puede usarse para muchos otros casos de uso, como creación de esquemas, creación de logos o incluso para la edición de diagramas UML.

4.3. Técnicas y Metodologías

Test Driven Development

Test Driven Development es una metodología de desarrollo agile en la que se implementan antes los tests unitarios que la funcionalidad de la aplicación. Sigue el esquema de tres fases **red**, **green**, **refactor**. La fase red indica que los tests deben fallar. La fase green indica que se debe implementar de forma correcta la utilidad para que los tests se ejecuten satisfactoriamente, y por último la fase refactor indica que implementación debe ser refactorizada para mejorar la mantenibilidad.

Metodología Scrum

Scrum es una metodología de gestión de proyectos a nivel de equipo. Scrum se centra en dividir las tareas del equipo en pequeñas tareas que puedan ser cumplidas durante un sprint.

No se ha seguido esta metodología al pie de la letra debido a que solo ha participado un único desarrollador en el proyecto, por lo que algunos conceptos como los roles de miembros o las reuniones diarias no se aplican. En este caso, se han realiza sprints con una duración de 2 semanas y reuniones para revisar los sprints al final de cada sprint. Se ha acompañado esta metodología con Zenhub 4.2 para definir las pilas de sprint y producto, estas pilas contienen las tareas que se van a realizar en el sprint actual y a lo largo del proyecto ordenadas por prioridad.

Aspectos relevantes del desarrollo del proyecto

5.1. Datasets

GTZAN Dataset

Las versiones iniciales del clasificador fueron desarrolladas sobre el conjunto de datos para clasificación de música GTZAN.

Este conjunto de datos [13] fue creado por George Tzanetakis en el año 2001 a partir de pequeños fragmentos de 30s extraídos de CDs, DVDs, Cassettes y grabaciones de la radio. Es uno de los conjuntos de datos más utilizados en los sistemas de reconocimiento de géneros musicales.

Este conjunto de datos está formado por 10 géneros, cada uno con 100 segmentos diferentes en formato .wav, mono, 16 bits de profundidad y una frecuencia de muestreo de 22050HZ.

- Blues
- Classical
- Country
- Disco
- Hip Hop
- Jazz
- Metal
- Pop

- Reggae
- Rock

Si bien este conjunto de datos es muy variado al contener 10 géneros muy diferentes entre sí y múltiples fuentes, presenta varios problemas:

- **Etiquetas incorrectas:** Esto es debido a que la clasificación de música es una tarea complicada incluso para el ser humano, siendo la precisión media del 70 %, como explora [14].
- **Distorsiones:** Si bien las distorsiones son una característica del conjunto de datos, es recomendable distorsionar la música mediante pre-procesamiento, ya que no todos los casos de uso requieren de un conjunto de datos con distorsiones. Los problemas relacionados con la calidad del conjunto de datos han sido explorados en [15].
- **Conjunto Reducido:** Únicamente disponemos de 100 segmentos de 30s por etiqueta.
- **Conjunto Obsoleto:** El conjunto fue creado a principios de los años 2000, por lo que no refleja los estilos de las dos últimas décadas de música.

GTZAN Extended

Como alternativa al conjunto de datos GTZAN, se decidió crear una versión extendida con 300 canciones por cada género. Este conjunto de datos fue creado a partir de playlists de Spotify creadas por la comunidad.

Si bien este nuevo conjunto de datos extendido incluye múltiples canciones nuevas, fue creado a partir de las canciones más populares de Spotify del siglo XX, por lo que no tenemos una variedad real de canciones.

Este conjunto de datos extendido ha sido publicado en Kaggle [16].

Ludwig Dataset

El Ludwig Dataset [17] es un conjunto de datos de música creado específicamente para este proyecto.

Existen una gran variedad de conjuntos de datos destinados al análisis de música, pero no hay ninguno que cumpla con todos los requisitos del proyecto.

- Géneros
- Subgéneros
- Estados de Ánimo
- Ficheros de Audio
- Canciones asociadas con Spotify
- Etiquetas Revisadas.

Para poder crear este conjunto de datos, ha sido necesario cruzar tres fuentes de datos distintas mediante el proceso de la figura 5.1

- **Discogs:** Es el Marketplace de música más importante del mundo, cada item está categorizado por género y subgéneros. Discogs dispone de una API pública que permite obtener datos sobre los distintos elementos.
- **AcousticBrainz y MusicBrainz:** MusicBrainz [18] es una base de datos colaborativa de música, no es muy completa pero cada elemento tiene un MBID que es compartido con AcousticBrainz y Discogs.

AcousticBrainz es un proyecto del Music Technology Group de la Universitat de Pompeu Fabra [19] que permite analizar y clasificar canciones a partir de la onda sonora.

Este proyecto incluye una API pública en la que se pueden consultar detalles de canciones, como estados de ánimo o etiquetas obtenidas a partir de múltiples clasificadores. Se puede consultar los datos de una canción a partir de su MBID.

- **Spotify:** La API pública de Spotify dispone de un endpoint de búsqueda a partir del nombre. La gran mayoría de las canciones de Spotify permiten descargar un fragmento de 30s con la parte más relevante de la canción.

Además de las canciones en formato .mp3, cada una de las canciones tiene dos ficheros de numpy (.npy) asociados con 10 segmentos de 3s con los MFFCs y Espectrogramas en Frecuencias de Mel extraídos mediante Librosa.

- Rock
 - Pop Rock

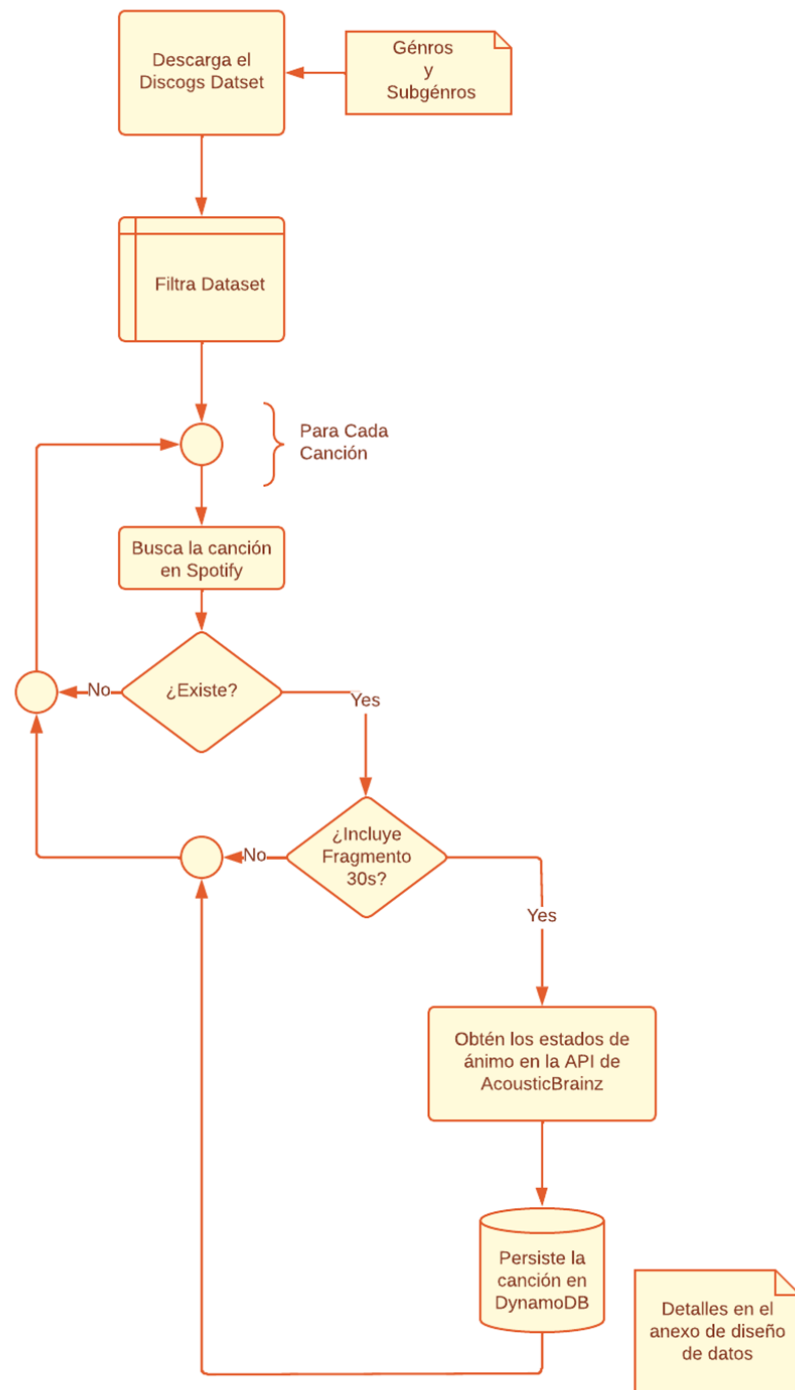


Figura 5.1: Proceso de obtención de los datos de Ludwig Dataset

- Hard Rock
- Prog Rock
- Rock Alternativo
- Goth Rock
- Art Rock
- New Wave
- Shoegaze
- Metal
 - Heavy Metal
 - Nu Metal
 - Death Metal
- Punk
 - Punk
 - Post Punk
- Blues
 - Country Blues
 - Electric Blues
- Latina
 - Reggaeton
 - Salsa
 - Flamenco
 - Reggae
 - Samba
 - Cubano
- Hip Hop
 - Trap
 - Pop Rap

- Instrumental
- Consciuous
- Gangsta
- Trip Hop
- Pop
 - Indie Pop
 - Europop
 - Ballad Pop
- Jazz
 - Jazz Contemporáneo
 - Swing
 - Soul Jazz
- Clásica
 - Clásica
 - Romántica
 - Barroca
 - Moderna
 - Ópera
- Electrónica
 - Ambient
 - Synth Pop
 - Disco
 - House
 - Drum n Bass
 - Downtempo
 - New Wave
 - Electro
 - Trip Hop

- Funk / Soul
 - Disco
 - R&B
 - Soul

Y los siguientes estados de ánimo

- Feliz / Triste
- Electrónica / Acústica
- Relajada / Fiesta
- Agresivo

Este conjunto de datos ha sido publicado en Kaggle [17].

Dataset Recomendaciones

Se ha generado un dataset de recomendaciones a partir de playlists aleatorias de Spotify extraídas del dataset oficial Millon Playlist Dataset [20]. El dataset aleatorio contiene 32.000 canciones¹⁴ con una popularidad de al menos el 30 %.

5.2. Unificación de las fuentes de datos

Uno de los mayores desafíos encontrados durante el diseño e implementación del Frontend ha sido la unificación de las distintas fuentes de datos en la capa de presentación.

Esto es debido a que ninguna de las APIs comerciales que han sido utilizadas permiten lanzar consultas o realizar búsquedas sobre elementos específicos.

Por ejemplo, la API de Spotify nos permite conocer todos los álbumes que ha guardado un usuario, pero no podemos limitar esta consulta a todos los álbumes guardados por el usuario de un artista en concreto.

Si queremos implementar una funcionalidad similar en nuestra aplicación necesitamos conocer de antemano todos los elementos del usuario, así como los distintos atributos de cada uno de los elementos.

¹⁴El nombre de cada canción contiene el id de Spotify, {id_spotify}.mp3

Spotify

Uno de los principales desafíos a la hora de integrar las distintas fuentes de datos con el frontend ha sido la API de Spotify. Esta API no devuelve los mismos datos a la hora de obtener un mismo recurso de varios endpoints distintos, por lo que los datos con los que trabajamos no son homogéneos.

Desde el punto de vista de nuestra implementación, este diseño de la API no nos beneficia ya que en algunos casos nos obliga a realizar varias peticiones sobre un mismo recurso para obtener todos sus atributos. Por otro lado, esta implementación tiene sentido desde el punto de vista del tamaño de las consultas, ya que si deseamos conocer las últimas 20 canciones que ha escuchado un usuario, es posible que no en todos los casos necesitemos conocer en que año se publicaron cada una de las canciones.

Tras realizar un estudio sobre la API, nos encontramos que todos los objetos que devuelve la API tienen tres tipos de objetos dependiendo de sus atributos:

- **Objeto Base:** Sus atributos son compartidos por todos los objetos de la API.
- **Objeto Simple:** Compuesto por los atributos que van a tener como mínimo todos los objetos de un tipo específico.
Por ejemplo, una canción además de tener sus atributos base, siempre va a incluir el objeto base del álbum en cualquier petición.
- **Objeto Completo:** Este objeto contiene todos los atributos posibles de un objeto.

El Frontend consume objetos completos, por lo que si al realizar una petición recibimos un objeto que no sea completo, debemos obtener el objeto completo de dichos elementos para poder trabajar con datos homogéneos.

LastFM

El principal inconveniente del diseño de la API de LastFM es que únicamente permite realizar una consulta por recurso, por lo que si queremos detallar los datos de 50 álbumes distintos debemos realizar 50 peticiones por separado.

Realizar un gran número de peticiones simultáneas es ineficiente por los siguientes motivos:

- Se realizan 50 consultas sobre la base de datos en vez de una única consulta.
- Se encolan múltiples peticiones HTTP.
- Se realizan múltiples consultas HTTP desde el cliente, saturando el único hilo de ejecución de JavaScript.

La solución propuesta ha sido acumular todas las peticiones individuales en bloques de 50 peticiones, delegando la tarea de lanzar 50 peticiones individuales al backend.

Por último, para evitar que estas peticiones en bloque sean extreamadamente lentas, el backend las realiza de forma concurrente para acelerar el proceso.

Analizador de Canciones

Una de las características del frontend es poder visualizar y poder filtrar por los resultados de un backend que analizar las canciones a partir de un fichero de audio correspondiente a un fragmento de la canción.

Este proceso es relativamente lento, ya que para cada canción es necesario realizar los siguientes procesos:

- Descargar el fichero de audio
- Convertir el audio a formato WAV.
- Muestrear el audio a una tasa de muestreo común.
- Extraer los MFCCs
- Inferir los coeficientes en un clasificador de estados de ánimo.
- Inferir los coeficientes en múltiples clasificadores de géneros y subgéneros.

De todos los procesos mencionados, los dos primeros pasos son los más lentos.

- La descarga está limitada por la latencia red de distribución de contenidos.
- La conversión a WAV es un proceso pesado

Para poder acelerar estos procesos se ha planteado la arquitectura CaaS, que permite distribuir las peticiones contra este analizador entre varios contenedores.

Como el proceso es extremadamente pesado, se ha implementado en la base de datos una tabla que almacena los resultados del analizador.

Etiquetas de Álbumes

La última fuente de datos del sistema es una tabla NOSQL que almacena las etiquetas personalizadas de los distintos usuarios.

Esta fuente de datos contiene datos dinámicos, que se pueden modificar tanto dentro del frontend, como desde otro cliente, por lo que deben mantenerse actualizados en todo momento.

Caché de fuentes de datos

Algunos de estos procesos son muy lentos, como el analizador de canciones o LastFM, y otros requieren de un gran número de peticiones, como Spotify o LastFM, por lo que el proceso de detallar al completo una canción o álbum puede afectar negativamente a la experiencia de usuario.

Para solventar este problema se ha implementado una caché de datos local que permita almacenar los datos estáticos de las canciones, álbumes y artistas en el propio navegador. Esta caché está implementada directamente en el frontend gracias a IndexedDB, una base de datos NOSQL presente en los estándares de W3C que está disponible en todos los navegadores modernos.

La caché local contiene tres tablas, Track, Album y Artist, que almacenan de forma conjunta todos los datos de cada una de las fuentes de datos. En este caso, una canción está compuesta por un álbum y uno o más artistas, y un álbum está compuesto por uno o más artistas.

Fachada de Datos

Una vez planteadas las distintas fuentes de datos y sus limitaciones, nos encontramos con que coordinar la obtención de estos datos puede ser una tarea muy compleja debido a los distintos estados de cada dato.

- Un dato puede estar incompleto

- Un dato puede estar almacenado localmente
- Una canción puede estar cacheada parcialmente (por ejemplo, puede estar cacheado el artista al que pertenece, pero no su álbum)
- Un dato contiene datos dinámicos

Para facilitar la obtención de datos se ha implementado una fachada que permite obtener todos los datos de manera eficiente.

Esta fachada permite obtener a partir de un ID o respuesta de Spotify, un objeto completo con todos los atributos de las distintas fuentes de datos.

5.3. Aprendizaje Automático

Datos y Votación

Para cada clasificador, se han dividido las canciones del dataset Ludwig 5.1 en dos subconjuntos aleatorios. Por un lado, tenemos el conjunto de entrenamiento que contiene el 70 % de las canciones que vamos a utilizar, este subconjunto será utilizado para ajustar el modelo. Por otro lado, tenemos un pequeño conjunto de validación con el 30 % de las canciones restantes, que utilizaremos para evaluar la precisión final de los modelos.

Para aumentar la variedad de datos, mejorar la generalización de la red, y evitar entradas muy grandes, se han dividido todas las canciones, de 30 segundos de duración, en segmentos de 3 segundos. Para cada uno de los segmentos se han extraído 32 MFCCs ¹⁵ 3.5, que serán utilizados como entrada de la red.

Por último, para mejorar la confianza de la predicción se realiza una votación para decidir la clase de la canción, esta votación tiene en cuenta cuales son las clases más populares entre todos los segmentos de 3s de la canción.

Clasificador de Géneros

Para el clasificador de géneros se ha utilizado una red neuronal convolucional basada en la arquitectura EfficientNetB0 [21] de Google. Se ha utilizado esta red como base debido a su capacidad de generalización para este problema.

¹⁵La entrada a la red es una matriz de 32×130

En este caso, la función de activación de la capa de salida es una función *softmax()*, que normaliza la salida para que solo se active una única neurona.

Inicialización de los Pesos

Durante el proceso de experimentación se ha detectado una mejora en la precisión del clasificador gracias a inicializar los pesos a partir de otra red basada en EfficientNetB0. Esta red ha sido entrenada con el conjunto de datos GTZAN Extended 5.1.

Para esta tarea se ha entrenado la red por completo y se ha reemplazado la capa de salida de 10 neuronas por la capa correspondiente a los géneros del Ludwig Dataset.

Clasificador de Estados de Ánimo

La clasificación de los estados de ánimo es un problema de clasificación multietiqueta, en el que una canción puede tener varios estados de ánimo.

En este caso, la clasificación de los estados de ánimo se ha realizado mediante un ensemble de tipo OVA 3.3, ya que el número de modelos base es relativamente pequeño, y podemos mejorar la confianza final del modelo. Cada modelo que forma el ensemble se especializa en detectar si una canción tiene un estado de ánimo en específico o no.

Por último, se han juntado todos los modelos en una única red neuronal para crear el ensemble, concatenando todas las salidas binarias en un único tensor de salida.

El modelo base utilizado para este ensemble es una red neuronal convolucional muy básica, en la que la capa de salida tiene una función de activación *sigmoide()* para que la salida sea binaria.

Clasificador de Subgéneros

Al igual que en el clasificador de estados de ánimo 5.3, la clasificación de subgéneros es un problema de clasificación multietiqueta, ya que una canción puede tener uno o más subgéneros. En este caso, debido a la gran cantidad de subgéneros que puede tener cada género, se ha entrenado un clasificador multietiqueta por cada uno de los géneros principales.

En este caso, la función de activación de la capa de salida ¹⁶ es la función

¹⁶La salida de la red tiene tantas neuronas como subgéneros.

sigmoide(), en la que cada neurona puede tomar un valor entre 0 y 1 para indicar si la canción se corresponde a una etiqueta o no.

5.4. Vecindad como Métrica de Similitud de Audio

Para crear un sistema de recomendación basado en contenidos se ha decidido utilizar un sistema de vecindad. Este sistema permite encontrar las canciones más cercanas o similares a una canción específica.

Para generar las instancias que van a ser comparadas, se ha decidido extraer el máximo número de datos de una canción mediante Librosa. Para resumir los resultados vectoriales o matriciales, como los MFCCs o los espectrogramas de mel 3.5, se han reemplazado por su media y varianza. Debido a las limitaciones del sistema, se han eliminado todos los datos cuyas operaciones de extracción tardaban más de 0.1s respecto a la media de operaciones¹⁷. Y para reducir aún más el tamaño del vecindario, realizó un estudio de la correlación de los datos mediante Knime 4.1.

Los datos finales utilizados para crear el vecindario son:

1. 32 medias y varianzas de los MFCCs 3.5
2. Tempo. El tempo permite conocer como de rápido va la música.
3. Media y varianza del tempo obtenidas de ventanas de tamaño *hop_length*.
4. 12 medias y varianzas del cronograma. El cronograma permite conocer los tonos más importantes del espectrograma¹⁸.
5. Media y varianza de los centroides del espectrograma
6. Media y varianza del ancho de banda del espectrograma. Se divide el espectrograma en ventanas de tamaño *hop_length*, el ancho de banda es la porción de frecuencias más representativa en la que se representa la señal, la frecuencia central del ancho es el centroide.
7. Media y varianza del contraste del espectrograma. Comparación entre la energía media (db) de cada ventana del espectrograma y la energía más elevada.

¹⁷Algunas de las características eliminadas son los tempogramas, centroides tonales, centroides espectrales o descomposición de espectrogramas

¹⁸Espectrograma en frecuencias de mel

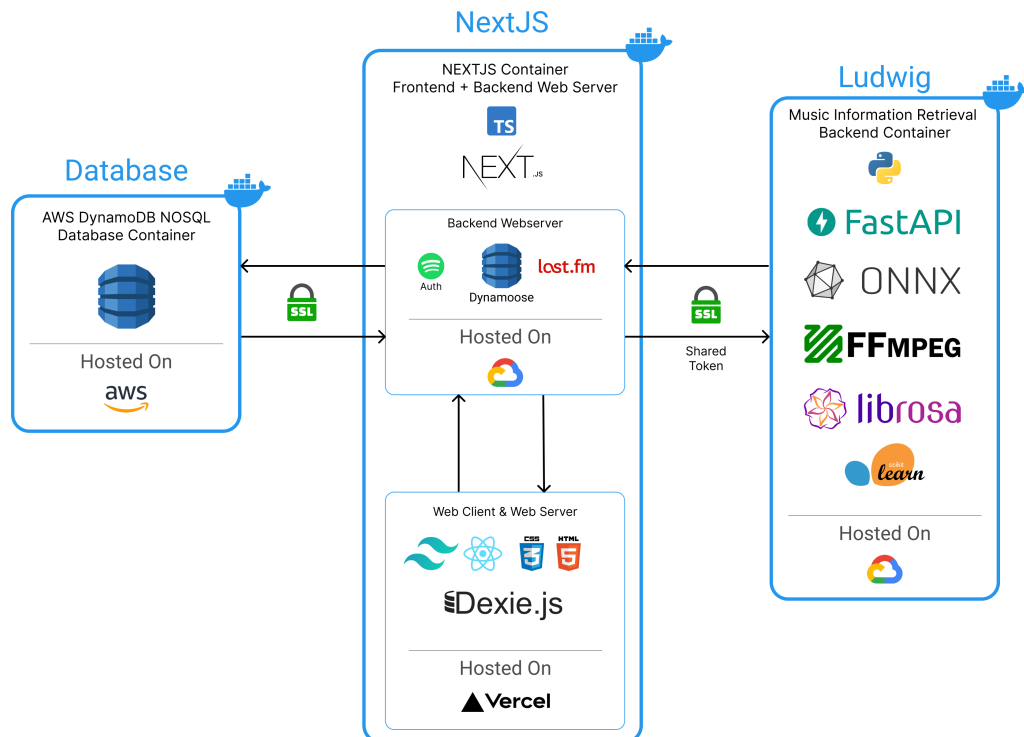


Figura 5.2: Arquitectura del Proyecto

Una vez creado el vecindario a partir del conjunto de datos 5.1, se normalizan los datos y se entrena un ball tree 3.4 para poder consultar los ids de Spotify de las canciones más similares de forma eficiente.

5.5. Arquitectura Contenedores

Se ha dividido la arquitectura en cuatro servicios, **frontend**, **backend principal o backend NextJS**, **base de datos** y **backend recuperación de información musical**¹⁹.

Estos cuatro servicios se han agrupado en tres contenedores Docker^{4.2} distintos interconectados, como muestra la figura 5.2.

El servicio puede levantarse en una sola máquina gracias a Docker Compose, se recomienda consultar el manual de instalación y despliegue en los anexos.

¹⁹También referido como Backend MIR o Backend Ludwig

Contenedor NextJS

El contenedor NextJS contiene el frontend y el backend principal, ya que ambos se ejecutan sobre un mismo servidor de NextJS.

El servicio de frontend se encarga del renderizado parcial en el servidor de las distintas páginas y componentes web, así como de la distribución de las páginas y ficheros estáticos.

El servicio de backend NextJS es una pequeña API REST que se encarga de gestionar operaciones con datos sensibles, como la autenticación del usuario, eliminación de la cuenta o la gestión de tokens JWT 4.1. Por otro lado, para reducir el acoplamiento del sistema, se ha decidido que este servicio sea el único servicio con acceso a la base de datos.

Esta API Rest ha sido configurada para permitir la política de seguridad CORS²⁰, por lo que el frontend y el backend pueden desplegarse por separado.

Contenedor DynamoDB

El contenedor DynamoDB contiene una instancia de DynamoDB Local 4.1, la única base de datos del sistema²¹. Este contenedor no está pensado para ser desplegado a producción, ya que una de las principales ventajas de DynamoDB es que es una base de datos administrada que se despliega en un cluster para mejorar su rendimiento. Para más información se recomienda consultar el apartado de despliegue 5.7.

Contenedor Ludwig MIR

El contenedor Ludwig MIR contiene una API de python que se encarga del preprocesamiento y análisis de canciones en cualquier formato de audio²². La política de seguridad de este contenedor obliga a todas las peticiones a portar un token de seguridad en la cabecera **Authentication**. Este token es un código secreto estático que debe ser compartido entre todos los servicios que desean interactuar con el contenedor. En este caso, únicamente el contenedor NextJS tiene acceso a este código, por lo que actúa como intermediario.

Debido al elevado coste de descargar, preprocesar y analizar canciones, se ha planteado el uso de una pequeña caché que almacene los resultados en

²⁰Intercambio de Recursos de Origen Cruzado, el frontend puede acceder a recursos del backend sin necesidad de compartir dominio.

²¹Sin tener en cuenta la caché del frontend con DexieDB

²²Gracias a FFMPEG

DynamoDB. La lógica de esta caché está gestionada por el backend de NextJS.

5.6. Pruebas e Integración Continua

Pruebas Unitarias

Se han realizado pruebas unitarias sobre los componentes de la capa de presentación, como los clientes REST, y clases y funciones auxiliares. Debido a la dificultad de renderizar componentes fuera de un navegador, ya que no todas las funcionalidades se pueden ejecutar fuera de un navegador sin el uso de mocks²³, se ha descartado realizar pruebas unitarias exhaustivas en la capa de la Vista.

Estas pruebas han sido realizadas mediante la biblioteca JEST 4.1

Pruebas Punto a Punto

Se han realizado pruebas automáticas y de sistema a la interfaz web, así como a los distintos endpoints de las APIS mediante Cypress 4.1. Estas pruebas se realizan directamente sobre la web desplegada. Los endpoints han sido implementados siguiendo la metodología TDD 4.3.

Integración Continua

Se ha programado mediante Github Actions 4.2 la ejecución de las distintas pruebas tras cada operación de push y merge al repositorio. Estas acciones permiten generar reportes sobre el estado del proyecto, y detectar los fallos en el momento en el que se actualiza el proyecto.

5.7. Despliegue Continuo

Google Cloud Run

Se han desplegado en contenedores de 4.2 los servicios NextJS Backend y Ludwig Backend. Estos contenedores se generan y despliegan automáticamente²⁴ tras cada push que modifique los backends en el proyecto.

²³Clases o funciones esqueleto que contienen todas las operaciones de la clase real, no todas las bibliotecas incluyen mocks.

²⁴Gracias a Google Container Registry, esta operación se puede delegar a Github Actions

El contenedor NextJS se ha alojado en máquinas con apenas 256MB de RAM y un núcleo, ya que este contenedor apenas realiza operaciones con un elevado coste de recursos.

Este contenedor tiene libertad para escalar automáticamente.

El contenedor Ludwig Backend se ha desplegado en máquinas con 2GB de RAM y un núcleo. Estos contenedores están pensados para procesar hasta 25 canciones al mismo tiempo, por lo que necesitan un mayor número de recursos para inferir con todas la redes neuronales.

Para reducir el consumo de recursos, se han cuantizado 3.2 los modelos a UINT8, reduciendo el consumo de memoria un contenedor de 6.8GB a menos de 2GB de RAM. Los consumos de memoria son tan elevados debido a que trabajamos con bloques de canciones para acelerar la inferencia.

Para acelerar el proceso de inferencia de un gran grupo de canciones, el orquestador está configurado para levantar el máximo número de contenedores en vez de encolar las peticiones de inferencia en grupos de 25 canciones. Se ha configurado un límite de 350 contenedores simultáneos antes de que el sistema empieza a encolar las peticiones de inferencia.

Vercel

El propio frontend de NextJS es transpilado, optimizado y desplegado por la plataforma Vercel 4.2. Vercel, a diferencia de Google Cloud Run, no admite contenedores personalizados, pero está diseñada para optimizar, desplegar y escalar proyectos de NextJS.

Se ha configurado Vercel para realizar un despliegue automáticamente por cada push y pull request que se realice en el repositorio del proyecto²⁵.

El backend de NextJS no se ha desplegado junto a su frontend por las limitaciones del plan gratuito de Vercel, ya que una petición al backend no puede durar más de 10s, un tiempo muy bajo para que Ludwig pueda inferir los bloques de 25 canciones.

Amazon Web Services

En este caso se ha utilizado el servicio DynamoDB 4.1 para la base de datos. Este servicio permite controlar y escalar la base de datos en al menos dos clusters, dividiendo los datos a partir del hash de los índices. Las tablas

²⁵Al igual que GCP, se puede automatizar mediante Github Actions

son gestionadas automáticamente gracias a Dynamoose, por lo que todas las actualizaciones se propagan a la base de datos de AWS.

Trabajos relacionados

6.1. Similitud de Audio Mediante Redes Neuronales Siamesas

En [5] presentan una aproximación mucho más interesante a la hora de calcular la similitud entre dos canciones usando redes neuronales siamesas. Este sistema utiliza dos redes neuronales exactamente iguales²⁶ Fig.6.1, con la que se obtiene una salida de 128 elementos. Esta salida es comparada mediante una función de pérdida que permite obtener la diferencia entre cada una de las entradas de la red. Se ha descartado esta aproximación de similitud por varias razones.

Entrenamiento: Para poder entrenar una red siamesa es necesario agrupar manualmente los datos según su similitud. Cada grupo deben contener N canciones similares y N canciones diferentes.²⁷

Inferencia: Para poder obtener la similitud de una canción es necesario comparar la salida de la red con todas las canciones mediante la función de comparación.

6.2. Servicios Similares

Spotify

Desde el inicio del proyecto, Spotify [22] ha añadido algunas funcionalidades que se asemejan a las implementadas en este proyecto. Por un

²⁶misma arquitectura y pesos

²⁷ N puede ser un número pequeño, de 4 o 5 canciones

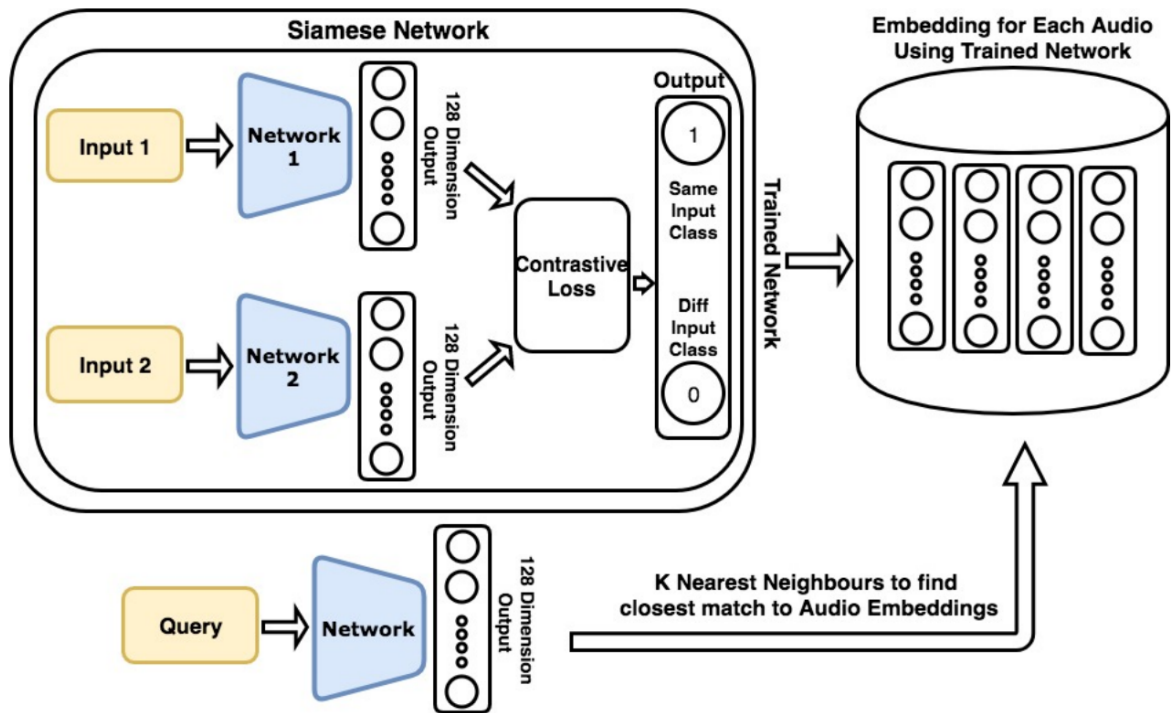


Figura 6.1: Sistema de Recomendación basado en RNS. Extraído de [5]

lado, Spotify ha añadido un sistema para rellenar playlists con canciones recomendadas, similar al sistema de recomendación de SpotMyFM. Por otro lado, Spotify crea periódicamente playlists personalizadas sobre un tema específico, como por el ejemplo **géneros**, **décadas**, **estados de ánimo** o **artistas**. Estas playlists están formadas por canciones de la biblioteca del usuario, así como recomendaciones.

Rate Your Music

Rate Your Music [23] (RYM) es una plataforma colaborativa destinada a la clasificación y calificación de música. RYM no tiene integración con Spotify, pero gracias a su base de datos se incluyen características similares a SpotMyFM, como el poder etiquetar álbumes y añadir filtros a nivel de etiquetas, géneros o fechas. Por otro lado, RYM incluye un sistema de recomendación basado en las calificaciones del usuario. Este sistema de recomendación colaborativo, al igual que el que se ha planteado para SpotMyFM, no funciona a tiempo real.

Tabla 6.1: Comparativa entre Obscurify y SpotMyFM

Funcionalidad	SpotMyFM	Obscurify
	Biblioteca	
Alcance del Análisis	Playlists	Canciones más escuchadas
	Canciones más escuchadas	
Géneros Favoritos	Si	Si
Valoración General de Popularidad	No	Si
Canciones/Artistas Menos Populares	Si (Métrica de Spotify)	Si (Métrica especial)
Canciones/Artistas Más Populares	Si	No
Artistas Favoritos	Si (Siempre)	Si (último mes)
Agrupamiento por Décadas	Si	Si
Recomendaciones	Si (Contenido)	Si (Colaborativo)
Estados de Ánimo	Si (análisis de la canción)	Parcial (Felicidad, Energía y Acústico)
Evolución de los géneros a lo largo del tiempo	Si	No
Filtros	Si	No
Creación de Playlists	Si (Cualquier Item)	Si (Solo recomendaciones)
Integración con LastFM	Si	No
Detalles de canciones / Artistas	Si	No

Obscurify

Obscurify [24] es una web de código abierto que permite consultar las estadísticas personales de Spotify de manera similar a SpotMyFM. La principal diferencia entre Obscurify y SpotMyFM es que Obscurify únicamente muestra las estadísticas de artistas y canciones más escuchados durante el **último mes**, **último medio año** y **desde el inicio**, mientras que SpotMyFM permite consultar las estadísticas de dichos intervalos, la biblioteca completa del usuario y playlists individuales.

Obscurify está centrada en el análisis de popularidad de una biblioteca, y realiza este análisis a partir de una métrica que tienen en cuenta la popularidad de una canción o artistas, así como su posición en el top personal del usuario. Obscurify, a diferencia de SpotMyFM, no está centrado en la gestión de la biblioteca.

La tabla 6.2 contiene una comparativa entre las funcionalidades de análisis de SpotMyFM y Obscurify.

Organize Your Music

Organize Your Music [25] (OYM) es una web que permite crear playlists de Spotify a partir de una selección de canciones. En este caso, la plataforma

OYM tiene integración con la API de Spotify, y está centrada en la creación de playlists a partir de otras playlists o la biblioteca del usuario, por lo que no incluye ningún tipo de detalles de las canciones. Esta plataforma clasifica las canciones a partir de sus géneros, duración, popularidad, décadas o duración, y permite seleccionar canciones individuales o grupos enteros. La principal diferencia entre el sistema de selección de SpotMyFM y OYM, es que OYM incluye un gráfico de dispersión de dos dimensiones con todas las canciones, en la que se puede seleccionar con un círculo las canciones que se encuentran en dicha gráfica. Los ejes del gráfico son personalizables, y el usuario puede escoger entre todas las características de las canciones, como la duración, popularidad, etc.

Conclusiones y Líneas de trabajo futuras

7.1. Conclusiones

Rendimiento de los clasificadores

La clasificación de música es un problema muy complejo, ya que cada artista tiene un estilo muy concreto, que puede estar influenciado por otros estilos y géneros, dificultando la tarea de clasificación de una pieza concreta. Por otro lado, la representación del audio, así como las técnicas para clasificarlo llevan varios años estancadas.

La OVA dedicada a la clasificación de estado de ánimo tiene una precisión media del 80 %, siendo esta una precisión muy similar a la que podemos encontrarnos en la literatura [26]. Por otro lado, el clasificador principal de géneros alcanza un 78 % de precisión en el conjunto de entrenamiento GTZAN Extended, superando a la precisión del ser humano, pero apenas alcanza el 59 % con el Ludwig Dataset, un conjunto de datos mucho más variado.

Limitaciones de depender de APIs

Si bien el uso de APIs pública es el único método para disponer de todos los datos y recursos del servicio, el no disponer de una base de datos completa ha causado un gran número de inconvenientes.

Diseño de Datos, tal y como se ha expuesto en el punto 5.2, ha sido

necesario unificar los datos y diseñar una caché local debido al elevado número de peticiones y heterogeneidad de las fuentes de datos.

Desconocimiento de Datos, debido a no tener acceso a todos los datos, no podemos disponer de conjuntos de datos más completos para los sistemas de recomendación, ya sea basada en contenido (etiquetas y similitud) o colaborativa.

Estabilidad del Sistema, debido a la elevada dependencia de tantas fuentes de datos ajenas al proyecto, si alguna de ellas falla, por ejemplo el inicio de sesión de Spotify, el frontend no se podrá utilizar.

Sistema de Recomendación Colaborativo basado en Playlists

Se ha implementado un prototipo de sistema de recomendación colaborativo basado en los contenidos de 200.000 playlists. Estas playlists de Spotify han sido recopiladas y anonimizadas por la propia compañía en [20]. Para la creación y evaluación de las recomendaciones se ha utilizado la biblioteca Scikit Suprise [11].

Este sistema de recomendación está basado en una puntuación a cada uno de los elementos de la playlist con una nota del 0 al 5, calculada a partir de la popularidad de la playlist, frecuencia de actualización, % de canciones del mismo artista o álbum, etc.

Las recomendaciones iban a integrarse con la plataforma como un sistema para completar la playlist de un usuario a partir de canciones ya existentes. Se ha descartado la idea por el elevado coste de añadir nuevos datos a una matriz dispersa, ya que es necesario volver a ajustar la matriz dispersa para cada una de las peticiones de recomendación.

Por otro lado se han intentado obtener las recomendaciones más cercanas a otra mediante la implementación de un algoritmo de vecindad de Surprise, pero debido a que esta implementación no usa matrices dispersas, el consumo de memoria necesario para obtener las canciones más cercanas es demasiado alto, por lo que la implementación de esta característica no es viable a nivel de recursos.

7.2. Líneas de Trabajo Futuras

Elastic Search

Se propone la creación de una base de datos con las instancias de los datos necesarios para el sistema de recomendación. Esta base de datos se debería ir expandiendo con nuevos contenidos a partir de las búsquedas de un usuario. Para realizar la operación de búsqueda de vecinos de manera eficiente, se propone el uso de Elastic Search[27], un motor de búsqueda avanzado para bases de datos que incluye un sistema de vecindad.

DynamoDB tiene integración con Elastic mediante OpenSearch, un fork de Elastic Search de AWS.

Ventanas Virtuales

Los navegadores webs no están diseñados para tener un gran número de elementos en el DOM, por lo que la simple tarea de mostrar un gran número de canciones, álbumes, playlists o artistas puede empeorar bastante el rendimiento, y con ello la experiencia de usuario.

Para evitar estos problemas de rendimiento se ha implementado un sistema de carga diferida (únicamente se muestran los elementos a medida que el usuarios se desplaza por la página) y de paginación, que limita el número de elementos por página para evitar sobrecargar al navegador.

Existe una solución mucho más elegante, la virtualización del HTML. Esta solución es muy utilizada en la web moderna por grandes plataformas como Twitter o Spotify. La virtualización consiste en añadir al DOM un número limitado de elementos (por ejemplo 20 tweets), y a medida que el usuario realiza scroll, desmontar del DOM los tweets antiguos y añadir los siguientes tweets en la parte inferior. Si bien esta técnica parece sencilla de implementar, aparece una pequeña limitación, y es mantener la barra de scroll en su posición correcta a medida que la ventana virtual avanza, así como cargar los elementos correctos al mover la barra de scroll a un punto intermedio.

La solución más sencilla consiste en utilizar elementos con altura fija, ya que así podemos añadir un espacio en blanco que rellene los elementos que se han desmontado, manteniendo la posición de la barra de scroll. Por ejemplo, si sabemos que hemos desmontado 100 elementos con una altura de 100px cada uno, debemos añadir un espacio en blanco de 10000px para mantener la posición del scroll, y si el scroll está a 2000px del inicio, sabemos que estamos frente al elemento número 20 de la lista.

En el caso de SpotMyFM, este tipo de cálculos no es posible realizarlo de forma eficiente ya que cada una de las tarjetas tiene una altura diferente, y el número de tarjetas por fila depende del ancho de la pantalla. Se plantea realizar un estudio que permita conocer los cambios necesarios para poder implementar esta técnica en el proyecto.

Expandir Dataset y Otras Representaciones

Si bien se han empleado un gran número de técnicas para mejorar el rendimiento del clasificador de géneros, se ha llegado a la conclusión de que los datos son el principal limitante.

Por un lado se ha planteado reemplazar los coeficientes cepstrales en las frecuencias de mel por espectrogramas en las frecuencias de mel. Este cambio de representación apenas mejora la confianza y tiene un elevado coste de memoria.

Se plantea el uso de un dataset mucho más grande, ya que la augmentación de datos apenas tiene impacto sobre el resultado como expone [28]. Se plantea el uso de Million Song Dataset [29] para intentar mejorar el rendimiento de los modelos.

Refactorización de Enums con Strings

En algunos puntos del frontend se han utilizado Enums o Records con strings para identificar selecciones (principalmente en los selectores y menús desplegables). Esto ha causado que algunos elementos no hayan podido ser internacionalizados correctamente debido a que los enums han sido declarados estáticamente fuera del alcance del hook²⁸ de traducción, y este hook únicamente funciona dentro de componentes ReactJS.

Otras Técnicas de Extracción de Información Musical

Existen muchas otras aproximaciones para trabajar con música, como la separación de las pistas que forman una canción [30], extraer las letras de canciones a partir del audio o extraer los sentimientos a partir de la letra para acompañar al clasificador de géneros, como explora [26]. Se plantea el estudio e implementación de algunas de estas técnicas para expandir el funcionamiento de la plataforma.

²⁸Una función específica de ReactJS que permite re-renderizar la interfaz de usuario para actualizar cambios de manera declarativa

Bibliografía

- [1] C. Mitcheltree and H. Koike, “White-box audio vst effect programming,” 02 2021.
- [2] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” 2016.
- [3] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” 2021.
- [4] P. Pawara, E. Okafor, M. Groefsema, S. He, L. R. Schomaker, and M. A. Wiering, “One-vs-one classification for deep neural networks,” *Pattern Recognition*, vol. 108, p. 107528, 2020.
- [5] P. Manocha, R. Badlani, A. Kumar, A. Shah, B. Elizalde, and B. Raj, “Content-based representations of audio using siamese neural networks,” 2017.
- [6] K. Jacobson, V. Murali, E. Newett, B. Whitman, and R. Yon, “Music personalization at spotify,” in *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys ’16, (New York, NY, USA), p. 373, Association for Computing Machinery, 2016.
- [7] K. Rao and M. k e, *Speech Recognition Using Articulatory and Excitation Source Features*. 01 2017.
- [8] O. R. developers, “Onnx runtime.” <https://onnxruntime.ai/>, 2021. Version: x.y.z.

- [9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [10] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” *Nature Methods*, vol. 17, pp. 261–272, 2020.
- [11] N. Hug, “Surprise: A python library for recommender systems,” *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020.
- [12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattemberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.
- [13] G. Tzanetakis and P. Cook, “Musical genre classification of audio signals,” *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.
- [14] M. Dong, “Convolutional neural network achieves human-level accuracy in music genre classification,” 2018.
- [15] B. L. Sturm, “The state of the art ten years after a state of the art: Future research in music information retrieval,” *Journal of New Music Research*, vol. 43, pp. 147–172, apr 2014.
- [16] “Gtzan extended wav | kaggle.” <https://www.kaggle.com/datasets/jorgeruizdev/gtzan-extended-wav/settings>. (Accessed on 06/21/2022).

- [17] “Ludwig music dataset (moods and subgenres) | kaggle.” <https://www.kaggle.com/datasets/jorgeruizdev/ludwig-music-dataset-moods-and-subgenres?select=mfccs>. (Accessed on 06/21/2022).
- [18] A. Swartz, “Musicbrainz: A semantic web service.,” *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 76–77, 2002.
- [19] H. S. J. U. Dmitry Bogdanov, Alastair Porter and S. Oramas, “The AcousticBrainz Genre Dataset: Multi-Source, Multi-Level, Multi-Label, and Large-Scale,” in *International Society for Music Information Retrieval Conference*, pp. 360–367, 2019.
- [20] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, “Recsys challenge 2018: Automatic music playlist continuation,” in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys ’18, (New York, NY, USA), p. 527–528, Association for Computing Machinery, 2018.
- [21] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” 2019.
- [22] “Spotify.” <https://spotify.com/>. (Accessed on 06/20/2022).
- [23] “Welcome! - rate your music.” <https://rateyourmusic.com/>. (Accessed on 06/20/2022).
- [24] “Obscurify music.” <https://obscurifymusic.com/home>. (Accessed on 06/20/2022).
- [25] “Organize your music.” <http://organizeyourmusic.playlistmachinery.com/>. (Accessed on 06/20/2022).
- [26] C. Laurier, J. Grivolla, and P. Herrera, “Multimodal music mood classification using audio and lyrics,” in *2008 Seventh International Conference on Machine Learning and Applications*, pp. 688–693, 2008.
- [27] “Búsqueda gratuita y abierta: Los creadores de elasticsearch, elk y kibana.”
- [28] R. L. Aguiar, Y. M. Costa, and C. N. Silla, “Exploring data augmentation to improve music genre classification with convnets,” in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

- [29] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, “The million song dataset,” in *ISMIR* (A. Klapuri and C. Leider, eds.), pp. 591–596, University of Miami, 2011.
- [30] A. Défossez, N. Usunier, L. Bottou, and F. Bach, “Music source separation in the waveform domain,” 2019.