



**UNIVERSIDAD DE CASTILLA-LA MANCHA**

**ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**TECNOLOGÍA ESPECÍFICA DE  
COMPUTACIÓN**

**TRABAJO DE FIN DE GRADO**

**Generación automática de playlist de canciones mediante técnicas  
de minería de datos**

**Miguel Ángel Cantero Vállora**

**Febrero de 2020**







**UNIVERSIDAD DE CASTILLA-LA MANCHA**

**ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**TECNOLOGÍA ESPECÍFICA DE  
COMPUTACIÓN**

**TRABAJO DE FIN DE GRADO**

**Generación automática de playlist de canciones mediante técnicas  
de minería de datos**

**Autor: Miguel Ángel Cantero Vállora**

**Directores: José Antonio Gámez Martín**

**Juan Ángel Aledo Sánchez**

**Febrero de 2020**



*A mis padres*



## **Declaración de Autoría**

Yo, Miguel Ángel Cantero Vállora con DNI 47072961-B, declaro que soy el único autor del Trabajo Fin de Grado titulado “Generación automática de playlist de canciones mediante técnicas de minería de datos” y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 12 de febrero de 2020

Fdo.: Miguel Ángel Cantero Vállora





## **Agradecimientos**

Ante todo, me gustaría agradecer a mis tutores, José Antonio y Juan Ángel, toda la ayuda que me han ofrecido durante la realización de este TFG, ya que sin ellos no habría sido posible su realización.

También me gustaría mostrar mis agradecimientos a la familia, amigos, compañeros de carrera y profesores que me han apoyado durante todos estos años.



## Resumen

Desde el auge de Internet y la aparición de los primeros reproductores de MP3, a finales de los años 90, el mundo de la música en formato digital ha ido evolucionando y ganando protagonismo hasta tal punto que en 2014 superó en ingresos al formato físico [1].

Actualmente, mediante una suscripción a un servicio de música en streaming, tenemos acceso a más de 50 millones de canciones (este es el caso de *Spotify*) [2]. Ante un catálogo de música tan extenso, surge la necesidad de ayudar al usuario a descubrir o sugerir contenido acorde con sus gustos musicales, ya que este no puede dedicar todo su tiempo a buscar entre todo el contenido disponible. Aparte, la habilidad de búsqueda disminuye constantemente, ya que el contenido aumenta de forma significativa conforme pasan los años. Para solucionar este problema, se hace uso de los sistemas de recomendación. Su influencia, presencia e importancia han ido en aumento a lo largo del tiempo, ya que nos ayudan a filtrar contenido interminable.

Este Trabajo de Fin de Grado tiene como objetivo construir un sistema para la creación o completado de playlists de canciones para el servicio de música en streaming *Spotify*. A partir de una playlist compuesta por un título y/o un conjunto inicial de canciones, nuestro sistema debe ser capaz de crear una lista desde cero (en el caso de que sólo se le proporcione el título) o completar una lista con canciones relacionadas a las facilitadas por el usuario y el título propuesto.



# Índice de Contenidos

<b>Agradecimientos .....</b>	<b>v</b>
<b>Resumen .....</b>	<b>vii</b>
<b>Índice de Contenidos.....</b>	<b>ix</b>
<b>Índice de Figuras .....</b>	<b>xi</b>
<b>Índice de Tablas.....</b>	<b>xiii</b>
<b>CAPÍTULO 1. Introducción .....</b>	<b>1</b>
1.1. Contexto .....	1
1.2. Motivación .....	4
1.3. Objetivos .....	5
1.4. Estructura de la memoria .....	5
<b>CAPÍTULO 2. Antecedentes y estado de la cuestión.....</b>	<b>7</b>
2.1. Sistemas de recomendación .....	7
2.2. Big Data .....	11
2.3. Servicios de streaming musical.....	12
2.4. Proyectos similares.....	15
<b>CAPÍTULO 3. Metodología y desarrollo .....</b>	<b>19</b>
3.1. Metodología .....	19
3.2. Desarrollo.....	21
<b>CAPÍTULO 4. Conjunto de datos .....</b>	<b>25</b>
4.1. Million Playlist Dataset.....	25
4.2. Búsqueda de playlists.....	29
4.3. Preparación del proceso de descarga.....	33
4.4. Descarga de playlists.....	34
4.5. Filtrado de playlists .....	39
4.6. Generación del conjunto de datos .....	48
<b>CAPÍTULO 5. Construcción del modelo de recomendación.....</b>	<b>55</b>
5.1. LightFM .....	55

5.2. Preprocesamiento .....	57
5.3. Definición del modelo .....	65
<b>CAPÍTULO 6. Experimentos y resultados .....</b>	<b>67</b>
6.1. Experimentos.....	67
6.2. Resultados .....	74
<b>CAPÍTULO 7. Conclusiones y propuestas.....</b>	<b>77</b>
7.1. Conclusiones .....	77
7.2. Propuestas.....	78
<b>Bibliografía.....</b>	<b>81</b>
<b>Contenido del CD .....</b>	<b>87</b>

## Índice de Figuras

Figura 1. Evolución de los ingresos por formato en los Estados Unidos .....	2
Figura 2. Evolución en detalle de los ingresos por formato en Estados Unidos.....	3
Figura 3. Cuota de mercado de los servicios de streaming de pago en 2019 .....	4
Figura 4. Esquema del funcionamiento de un sistema de recomendación híbrido.....	9
Figura 5. Comparación gráfica entre filtrado por contenido y filtrado colaborativo.....	10
Figura 6. Muestra de datos generados en 60 segundos durante 2019.....	12
Figura 7. Ejemplo de radios ofrecidas por Amazon Music .....	13
Figura 8. Ejemplo de una playlist creada por Spotify.....	14
Figura 9. Captura de la página web del proyecto MagicPlaylist .....	15
Figura 10. Ejemplo de una recomendación con Playlist Generator.....	16
Figura 11. Ejemplo de uso del proyecto Spotalike .....	17
Figura 12. Fases del modelo de referencia CRISP-DM.....	19
Figura 13. Libreta de ejemplo ejecutada en Jupyter Notebook .....	22
Figura 14. Ejemplo del campo “info” de un archivo JSON del MPD .....	26
Figura 15. Ejemplo de una playlist contenida en el MPD .....	28
Figura 16. Fragmento del dataframe empleado para el filtrado de playlists.....	42
Figura 17. Alfabeto de caracteres válidos para los títulos de las playlists.....	45
Figura 18. Muestra del dataframe que contiene la información de las playlists.....	60
Figura 19. Muestra del dataframe que contiene las pistas que pertenecen a una playlist....	60
Figura 20. Muestra del dataframe que contiene la información de las pistas .....	60
Figura 21. Fragmento el diccionario de emoticonos.....	61
Figura 22. Muestra del dataframe que contiene las etiquetas de los nombres de playlists..	62
Figura 23. Ejemplo de matriz de interacción .....	64
Figura 24. Ejemplo de matriz de pesos de las interacciones .....	64
Figura 25. Parámetros por defecto para crear el modelo LightFM.....	65
Figura 26. Resultados de canciones similares para "Toxic" .....	68
Figura 27. Resultados de canciones similares para "Cool" .....	68
Figura 28. Resultados de canciones similares para "Six Feet Under" .....	68
Figura 29. Resultados para la playlist conocida nº1 .....	70
Figura 30. Resultados para la playlist conocida nº2 .....	70

Figura 31. Resultados para la playlist conocida nº3 .....	71
Figura 32. Resultados para la playlist aleatoria nº1 .....	72
Figura 33. Resultados para la playlist aleatoria nº2 .....	72
Figura 34. Resultados para la playlist aleatoria nº3 .....	73
Figura 35. Ejemplo de modelo secuencial .....	79



## Índice de Tablas

Tabla 1. Especificaciones técnicas del equipo principal.....	23
Tabla 2. Especificaciones técnicas de máquina virtual Serie B.....	24
Tabla 3. Especificaciones técnicas de máquina virtual 'Serie Fsv2' .....	24
Tabla 4. Resultados del proceso de búsqueda.....	32
Tabla 5. Resultados del proceso de preparación de descarga .....	34
Tabla 6. Resultados del proceso de descarga.....	38
Tabla 7. Valores perdidos para el dataframe de playlists descargadas .....	41
Tabla 8. Resultados obtenidos tras el proceso de filtrado.....	48
Tabla 9. Métricas para las playlists seleccionadas.....	71
Tabla 10. Métricas para las playlists aleatorias .....	73
Tabla 11. Resultados obtenidos en el conjunto de prueba .....	75



# CAPÍTULO 1. INTRODUCCIÓN

---

## 1.1. CONTEXTO

La forma en la que consumimos música ha ido evolucionando junto a la aparición de nuevas tecnologías. Antes del boom de Internet y la música en formato digital, los medios más frecuentes para consumir música eran las emisoras de radio y las colecciones privadas de música en soporte físico (vinilo, casete, CD, ...). A principios de los años 80, la televisión se convierte en otro medio gracias a la aparición de canales musicales como *MTV*, los cuales propiciaron el auge de la industria del videoclip.

La radio y televisión eran las formas más frecuentes de descubrir nuevo contenido, donde también se podía conocer qué canciones eran las más populares en un momento dado gracias a las listas de éxitos (ya sea por petición de los oyentes/espectadores o por información sobre las ventas de música en formato físico).

A finales de los años 90, con el auge de Internet y la aparición de los reproductores de MP3, la música en formato digital cobra protagonismo ya que el usuario es capaz de almacenar miles de canciones en un único dispositivo frente a los típicos formatos físicos, que eran capaces de almacenar algo más de 1 hora de música (alrededor de 15 canciones). Con la aparición de la música en formato digital, surgen dos nuevas vías de acceso: las tiendas musicales online, con *iTunes* (2003) como referente, y las redes de intercambio de archivos como *Napster* (1999), que generaron numerosas protestas entre las empresas discográficas por los derechos de autor. Paralelamente, con la aparición de las redes sociales, como *Facebook*, *Twitter*, *Last.fm*, ..., los usuarios encuentran una nueva forma para compartir sus gustos musicales y las canciones que suelen escuchar.

Hasta este momento, el usuario seguía limitado a su colección privada de música. Todo esto cambió con la llegada de los servicios de música en streaming, cuyo principal

## CAPÍTULO 1. Introducción

protagonista fue *Spotify* (2006). Otro medio que también ha adquirido importancia en estos últimos tiempos ha sido el de los servicios de alojamiento de video, como *YouTube* (2005), donde podemos encontrar un enorme catálogo de videoclips y otros tipos de contenidos musicales bajo demanda.

En la actualidad, y tomando como referencia el mercado estadounidense, el formato digital supone la mayor parte de los beneficios en la industria musical, mientras el formato físico sigue en decadencia. En la Figura 1 podemos ver la evolución de los ingresos de los formatos físico y digital.

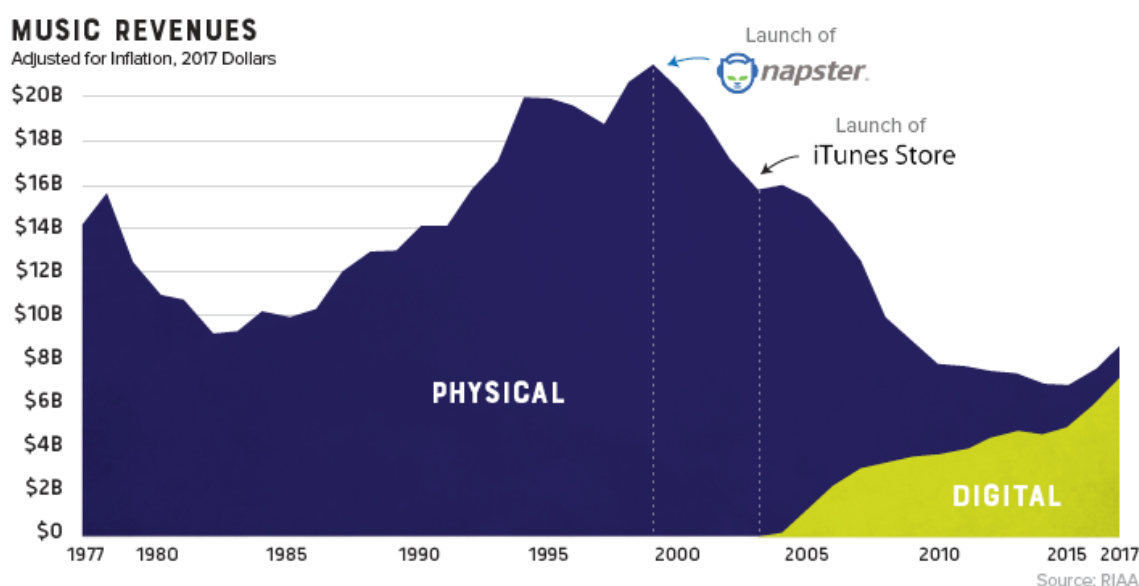


Figura 1. Evolución de los ingresos por formato en los Estados Unidos [34]

Si vemos la evolución teniendo en cuenta los distintos tipos de soportes que existen en el formato físico y las diferentes vías que ofrece el formato digital, como podemos ver en la Figura 2, comprobamos que desde 2014 el consumo de música mediante los servicios de streaming superan al de ventas de canciones en formato MP3, suponiendo más de la mitad de los ingresos [1].

Existen dos modelos de suscripción en estos servicios, el gratuito (soportado mediante anuncios publicitarios) y el de pago o *premium*, siendo este último el más

empleado. En la actualidad, *Spotify*, *Apple Music* y *Amazon Music* son los servicios con más suscriptores, tienen juntos más del 65% en cuota de mercado (Figura 3).

Gracias a los servicios de streaming, el usuario es capaz de tener a su alcance un catálogo conformado por más de 50 millones de canciones [2]. Ante tal cantidad de contenido, surge la necesidad de facilitar al usuario recomendaciones musicales para ayudarlo a descubrir nuevo contenido, ya sea mediante la recomendación de artistas o canciones similares a las que escucha con frecuencia, empleando las conocidas *radios* de los servicios de streaming, o recomendando nuevas canciones a añadir a las *listas de reproducción* o *playlists* que éste crea en su biblioteca. Ante tal necesidad, es donde entran en acción los sistemas de recomendación.

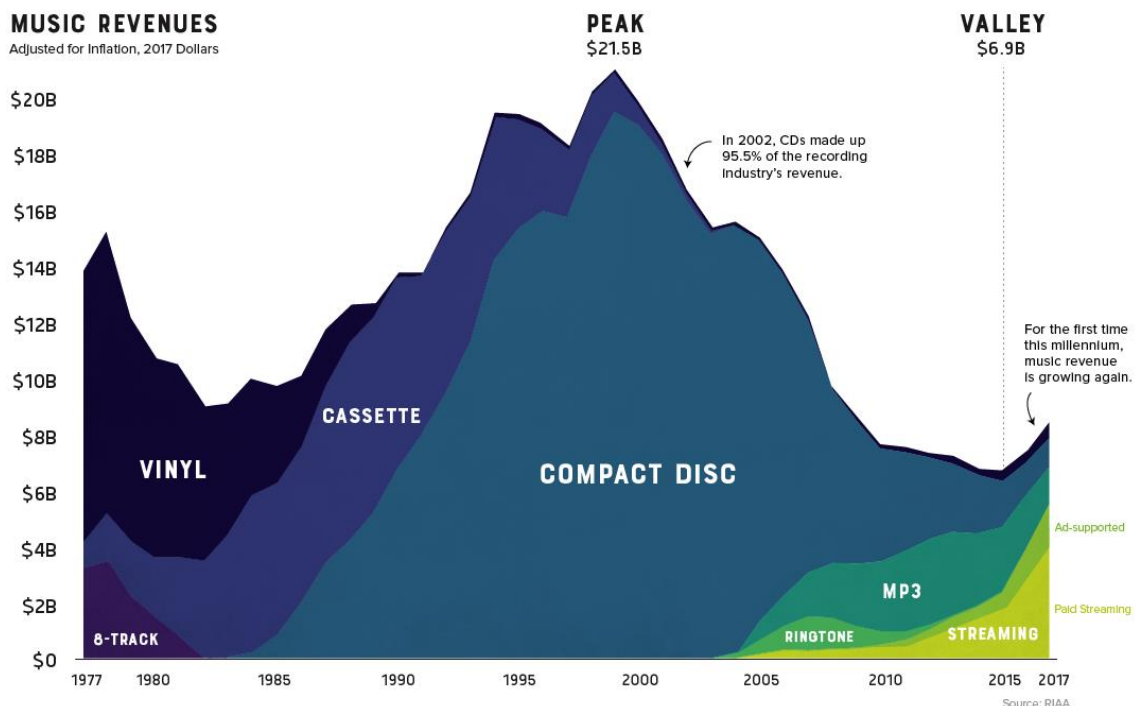


Figura 2. Evolución en detalle de los ingresos por formato en Estados Unidos [34]

Los sistemas de recomendación llevan mucho tiempo entre nosotros, pero gracias al auge del comercio electrónico, servicios de streaming, etc., es cuando han cobrado una mayor importancia. En el siguiente capítulo veremos con más detalle qué es un sistema de recomendación y los distintos tipos que existen.

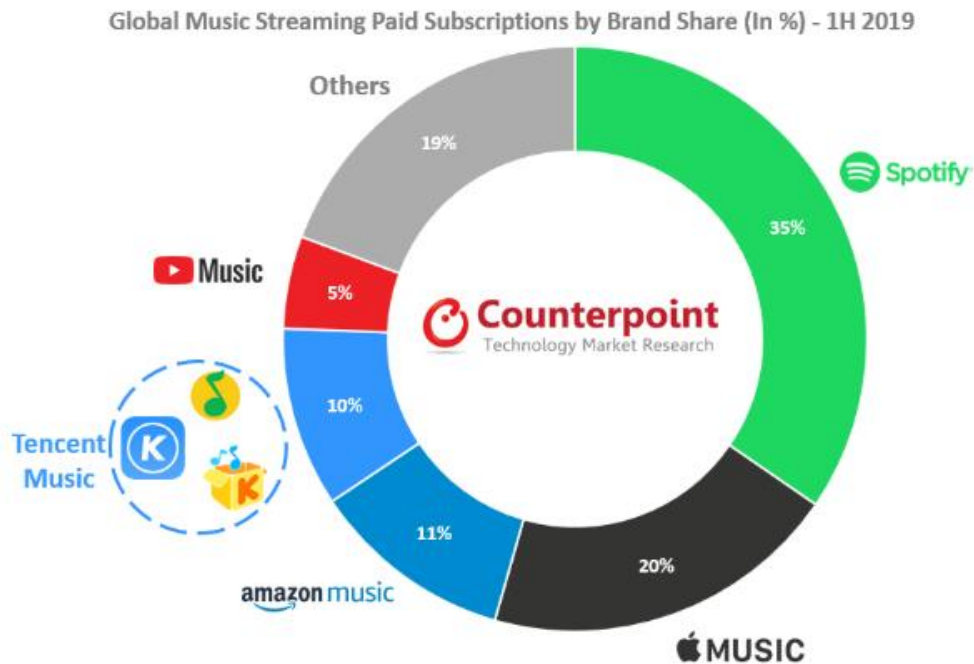


Figura 3. Cuota de mercado de los servicios de streaming de pago en 2019 [35]

### 1.2. MOTIVACIÓN

La principal motivación de este TFG surge por la curiosidad de crear un sistema de recomendación junto a mi afición por el mundo de la música. Su propósito es intentar completar playlists de canciones de forma alternativa a como lo hace *Spotify*.

Otro de los motivos es la actual importancia que están cobrando los sistemas de recomendación en el mundo del comercio electrónico, plataformas de video bajo demanda, agregadores de noticias, servicios de streaming musical, búsqueda de resultados, recomendación de amistades en redes sociales, etc.

Por último, otra de las razones que me animaron a llevar a cabo este proyecto fue la competición *ACM RecSys Challenge* [3], la cual organiza cada año un reto junto a una empresa colaboradora, quien proporciona la temática del reto. En 2018, *Spotify* colaboró en esta competición y el reto fue la continuación automática de playlist musicales [4].

## 1.3. OBJETIVOS

En esta sección vamos a explicar los principales objetivos que queremos lograr durante la elaboración de este trabajo.

### 1.3.1. OBJETIVO PRINCIPAL

El principal objetivo de este trabajo es construir un sistema de creación o continuación de playlists a partir de un nombre y/o una lista de canciones. Nuestro sistema también deberá ser capaz de solventar el problema del “arranque en frío”, cuando en la playlist que se le facilita sólo se dispone del título de esta.

### 1.3.2. OBJETIVOS SECUNDARIOS

Aparte del objetivo principal que hemos mencionado anteriormente, durante la realización de este trabajo también nos proponemos alcanzar los siguientes objetivos:

- Creación de un proceso de búsqueda de playlists.
- Implementación de un proceso de descarga para las playlists de canciones.
- Creación de un proceso de filtrado para las playlists obtenidas.
- Construcción de un conjunto de datos.
- Diseño de un modelo capaz de realizar recomendaciones sobre las playlists obtenidas.
- Estudio de los resultados obtenidos.

## 1.4. ESTRUCTURA DE LA MEMORIA

La manera en la que esta memoria está estructurada se resume en los siguientes puntos:

## CAPÍTULO 1. Introducción

- **Capítulo 1 – Introducción:** Contiene información acerca del tema elegido y la estructura de este trabajo, así como los motivos que han llevado a su elección.
- **Capítulo 2 – Antecedentes y estado de la cuestión:** En este capítulo hablaremos de los sistemas de recomendación y los tipos que existen, de los servicios de streaming disponibles y de las funcionalidades que ofrecen para descubrir contenido musical. También mencionaremos proyectos similares que existen en la actualidad.
- **Capítulo 3 – Metodología y desarrollo:** Explicación de la metodología utilizada para realizar el trabajo junto a las herramientas software y hardware empleadas.
- **Capítulo 4 – Conjunto de datos:** Detalle del proceso completo de creación del conjunto de datos: búsqueda, descarga, filtrado, etc.
- **Capítulo 5 – Construcción del modelo de recomendación:** Contiene la explicación del modelo a usar y el motivo que nos ha llevado a su elección. También detallamos el preprocesamiento de los datos que hemos realizado y los y definiremos los parámetros que usaremos en el modelo.
- **Capítulo 6 – Experimentos y resultados:** Detalle de los experimentos realizados para comprobar las predicciones del modelo entrenado y resultados tras emplear el modelo en el conjunto de prueba.
- **Capítulo 7 – Conclusiones y propuestas:** Contiene las conclusiones que se han obtenido tras concluir el trabajo junto a una serie de propuestas que se desean realizar en un futuro.



## CAPÍTULO 2. ANTECEDENTES Y ESTADO DE LA CUESTIÓN

---

Tras estudiar la evolución de nuestros hábitos de consumo de contenido musical y ver la importancia que tienen los servicios de música en streaming en la actualidad, es el momento de hablar de los sistemas de recomendación, los distintos tipos de sistemas que existen y el *Big Data*. También hablaremos de algunas de las funcionalidades que ofrecen los servicios de streaming musical más populares que permiten al usuario descubrir nuevas canciones. Por último, haremos un breve repaso a proyectos similares que existen en la actualidad.

### 2.1. SISTEMAS DE RECOMENDACIÓN

Cuando hablamos de sistemas de recomendación, nos referimos a aquellos sistemas que son capaces de predecir el grado de preferencia que un usuario tendrá para un conjunto de items, recomendándole aquellos que resulten de más interés para él.

En la actualidad, los sistemas de recomendación adquieren un papel fundamental, ya que gracias a Internet las personas se encuentran con demasiadas opciones para elegir. Por ejemplo, en un videoclub nos encontramos un número limitado de películas en relación con el tamaño del establecimiento. Por el contrario, si nos fijamos en *Netflix*, existe un número elevado de películas que podemos ver en línea. Con esto surge el problema de que las personas, ante tal cantidad de contenido, tienen dificultades para seleccionar y visualizar aquellas películas que pueden ser de su interés. En este escenario es donde los sistemas de recomendación ayudan a los usuarios a filtrar el contenido disponible de acuerdo con sus intereses y/o visualizaciones anteriores [5].

## CAPÍTULO 2. Antecedentes y estado de la cuestión

En un sistema de recomendación, los datos de los que hacemos uso pueden ser de dos tipos [6]:

- Información característica: Información sobre los items a recomendar (palabras clave, categorías, ...) y usuarios (preferencias, perfiles, ...).
- Interacciones usuario-item: Información sobre calificaciones, número de elementos adquiridos (compras), “likes”, etc.

### 2.1.1. TIPOS DE SISTEMAS DE RECOMENDACIÓN

Existen tres tipos importantes de sistemas de recomendación: los sistemas de *filtrado colaborativo*, los de *filtrado basado en contenido* y los sistemas de recomendación *híbridos*.

#### FILTRADO COLABORATIVO

Este método de filtrado se basa en recopilar y analizar información relacionada con los comportamientos del usuario, como actividades, preferencias, etc., y realizar una predicción de lo que le puede gustar en función de su similitud con otros usuarios [6].

Ejemplo: Si un usuario *A* valora positivamente las películas *John Wick*, *Misión Imposible* y *Jungla de Cristal*, y un usuario *B* valora positivamente las películas *Misión Imposible*, *Jungla de Cristal* y *El Caso Bourne*, entonces tienen intereses similares y al usuario *A* podría gustarle la película *El Caso Bourne* y al usuario *B* podría gustarle la película *John Wick*.

#### FILTRADO BASADO EN CONTENIDO

En este tipo de filtrado, nos centramos en las características o atributos de los items. En los sistemas de recomendación basados en contenido, las palabras clave se usan para describir los elementos. Además, también se crea un perfil de usuario para indicar el tipo de ítem que podría gustarle [6].

Ejemplo: Si un usuario ha valorado positivamente *Star Trek*, *Matrix* y *Blade Runner*, sabemos que le gustan las películas cuyo género es *ciencia ficción*. Por lo tanto, las recomendaciones se adaptarán para mostrar películas de dicho género.

En la Figura 5 podemos ver de forma gráfica una comparación de cómo funcionan los sistemas de recomendación basados en filtrado colaborativo y los sistemas de recomendación basados en contenido.

## SISTEMAS DE RECOMENDACIÓN HÍBRIDOS

Se basa en combinar los métodos de filtrado colaborativo y filtrado basado en contenido. Investigaciones recientes muestran que combinar ambos tipos de sistemas es más efectivo y se proporcionan recomendaciones más precisas [6].

En la Figura 4 podemos ver de forma gráfica cómo funciona un sistema de recomendación híbrido.

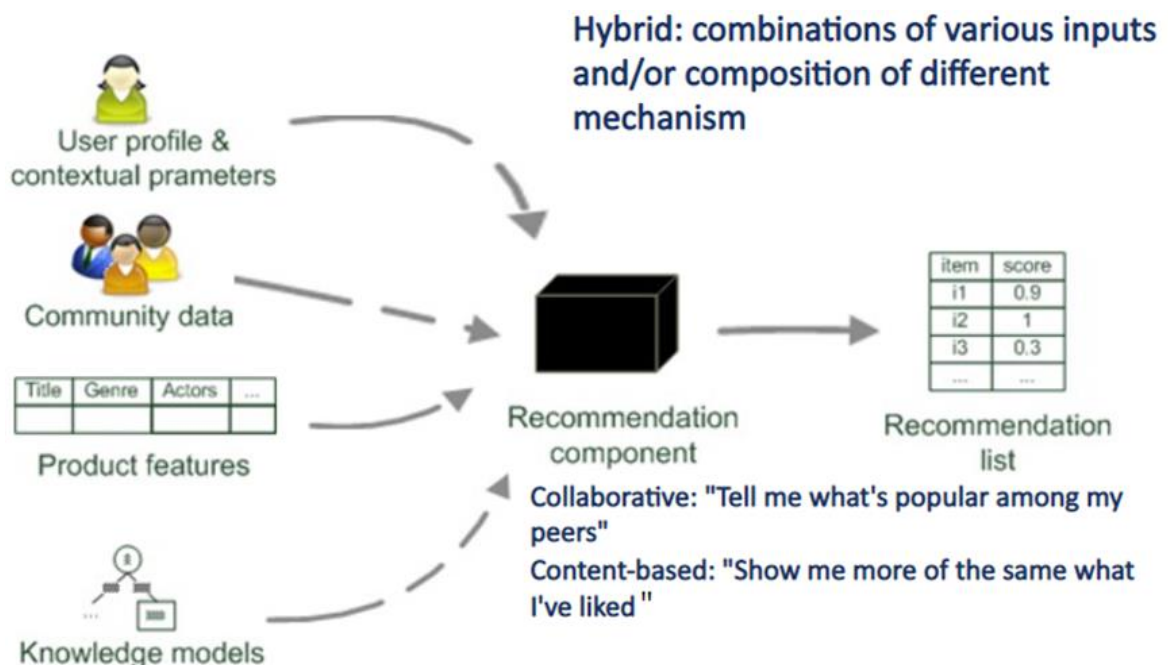


Figura 4. Esquema del funcionamiento de un sistema de recomendación híbrido [36]

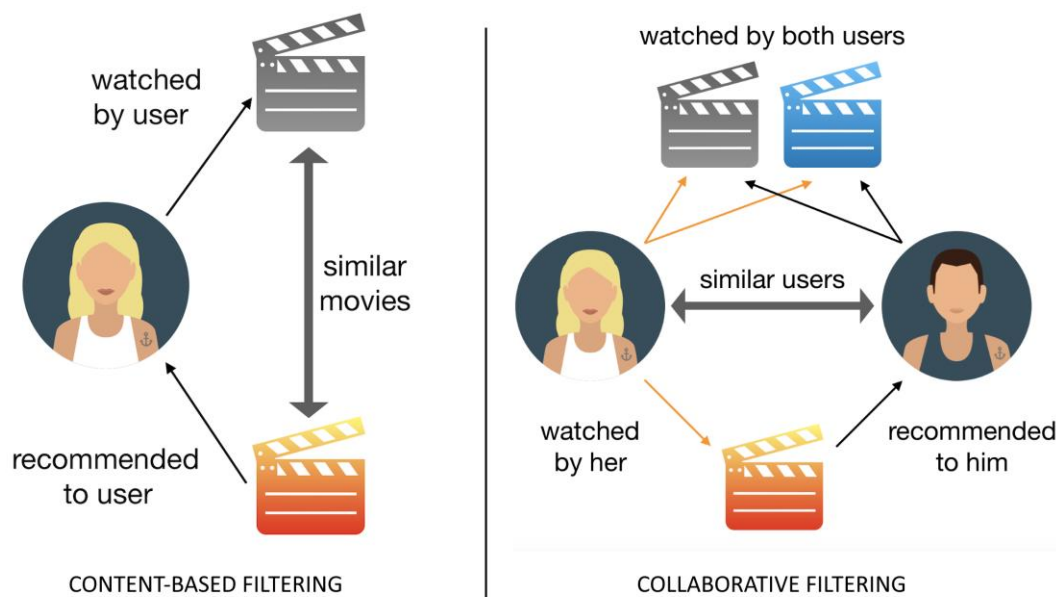


Figura 5. Comparación gráfica entre filtrado por contenido y filtrado colaborativo [37]

### 2.1.2. EL PROBLEMA DEL “ARRANQUE EN FRÍO”

El problema del *arranque en frío* (o “cold start” en inglés), es aquel que se da cuando nos encontramos con un nuevo usuario en el sistema del que no tenemos información sobre sus gustos para realizar una recomendación. Este problema también aparece con los nuevos items, ya que no existe interacción alguna con los usuarios que tenemos almacenados.

Éste lo podríamos solucionar mediante el uso del filtrado colaborativo. Si nuestro sistema ha requerido de un registro del nuevo usuario, según sus datos (edad, país, sexo, ...) podemos buscar usuarios con características similares y realizar una recomendación. Otra técnica que suele emplearse es haciendo uso del contexto del usuario. Por ejemplo, si por geolocalización hemos averiguado que el usuario reside en España, podemos recomendarle los items más relevantes en ese momento para dicho país, o atendiendo a otros factores como el tiempo (hora actual, época, estación, ...). En el caso del comercio electrónico, por ejemplo, si llega a nuestro sistema un nuevo usuario en épocas navideñas, el sistema puede hacer recomendaciones de artículos navideños (ya que existe una gran probabilidad de que esté buscando un regalo, elementos decorativos, ...) [7].

## 2.2. BIG DATA

Como hemos mencionado anteriormente, el número de datos está aumentando de forma drástica durante estos últimos años. En la Figura 6 puede verse un ejemplo, de 2019, con la cantidad de datos que se generan cada 60 segundos. Ante tal volumen de datos, nos encontramos con la necesidad de emplear un sistema para la gestión y análisis de grandes volúmenes de datos que no podemos tratar de forma convencional mediante software y hardware [8]. En esta situación es cuando hablamos de *Big Data*.

Las principales características que posee el *Big Data*, y que se conocen como las 3 “V” (en algunas ocasiones se habla de 5), son las siguientes:

- Volumen: Cuando escuchamos el término *Big Data*, lo relacionamos con un tamaño de datos que es enorme. No hay establecida una cantidad mediante la cual podamos considerar que estamos hablando de *Big Data*, aunque cuando el tamaño de los datos con los que estamos trabajando no puede ser gestionado por las capacidades de un solo ordenador, es cuando nos encontramos frente a un problema de este tipo [9].
- Velocidad: Otro factor importante en el cual *Big Data* difiere frente a otros sistemas de datos es la velocidad que tarda en moverse la información por el sistema. Es común encontrarnos con sistemas que obtienen información de diferentes fuentes y se requiere que los datos sean procesados en tiempo real [10].
- Variedad: Los datos con los que nos encontramos en un problema de *Big Data* son completamente heterogéneos, ya que pueden encontrarse en formatos como texto, video, imágenes, datos numéricos, etc. [11].

Otras características que también posee el *Big Data* son la veracidad, variabilidad y valor:

- Veracidad: Saber si los datos disponibles provienen de una fuente fiable es de suma importancia antes de descifrar e implementar *Big Data* [11].

- Valor: *Big Data* no es solo un conjunto de nuevas tecnologías, sino que es una combinación de negocio y analítica para obtener valor de los datos (convertirlos en información y conocimiento).

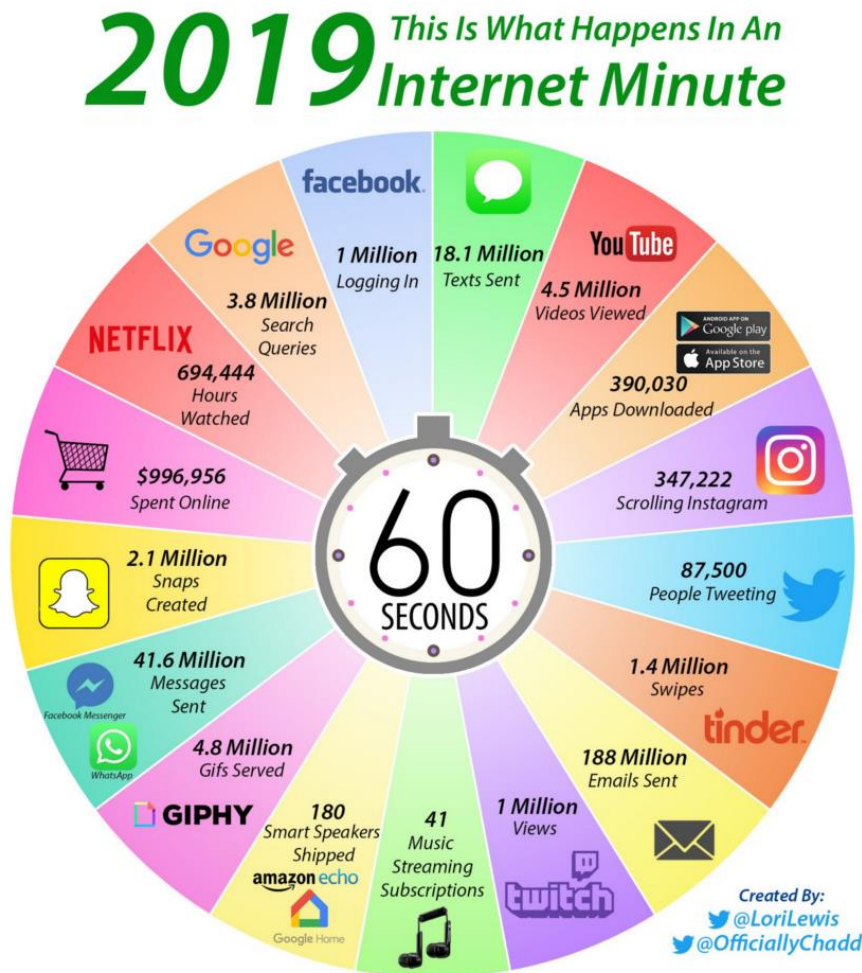


Figura 6. Muestra de datos generados en 60 segundos durante 2019 [38]

### 2.3. SERVICIOS DE STREAMING MUSICAL

En el Capítulo 1 hemos introducido los servicios de música en streaming, junto a las modalidades de suscripción que existen y los proveedores que tienen mayor cuota de mercado. En esta sección, nos vamos a centrar en aquellas funcionalidades que ofrecen ayuda al usuario para encontrar nuevo contenido musical.

### 2.3.1. FUNCIONALIDADES PARA DESCUBRIR CONTENIDO

Los servicios como *Spotify*, *Apple Music* y *Amazon Music*, que como hemos visto son los más destacados, ofrecen funcionalidades muy similares entre ellos. Si nos fijamos en aquellas que el usuario utiliza para añadir contenido a su colección, las más empleadas/conocidas son las siguientes: *radios*, *listas de éxitos* y *playlists*.

#### RADIOS

Las radios, también conocidas como emisoras, son una herramienta que se basa en que cuando un usuario está reproduciendo una canción, un álbum o un artista, al iniciar este servicio se reproducen canciones relacionadas/similares al tipo de contenido elegido de forma aleatoria (varía en cada ejecución). Por ejemplo, si un usuario inicia la radio de la cantante *Mariah Carey*, se reproducen canciones de esta artista junto a canciones de cantantes que son similares a ella. Otro ejemplo, en el caso de elegir una canción, la radio reproducirá canciones que sean similares o estén relacionadas con la que hemos indicado, (ya sea por el género musical, época, ritmo, artista similar, ...).

Algunos servicios también crean sus radios, pero en este caso pueden ser de una temática establecida. En Figura 7 podemos ver un ejemplo de radios creadas por *Amazon Music*.

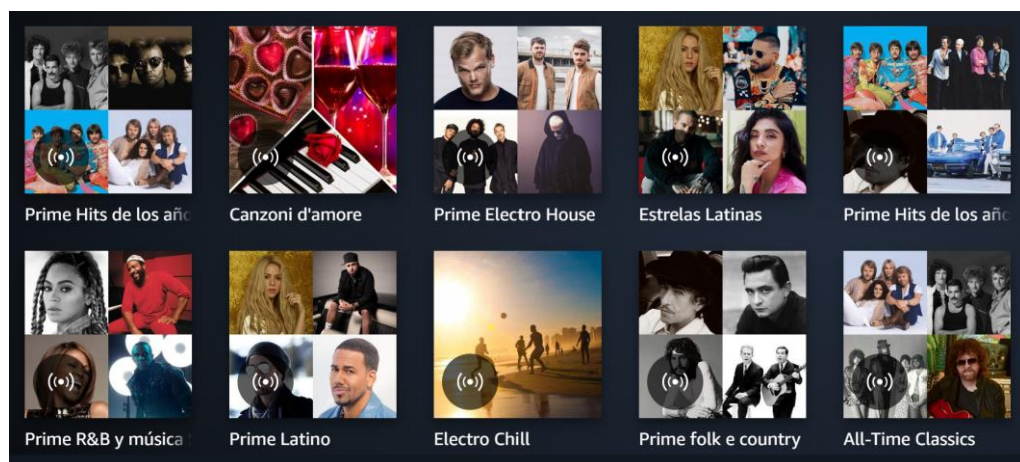


Figura 7. Ejemplo de radios ofrecidas por Amazon Music



### LISTAS DE EXITOS

Las listas de éxitos musicales llevan con nosotros varias décadas. En los servicios de música en streaming, nos ofrecen listas de éxitos relacionadas con el número de reproducciones que tienen las canciones en ese momento, ya sea a nivel mundial o por país. De igual manera que se hace con el número de reproducciones, si el servicio ofrece la posibilidad de comprar contenido, se ofrecen listas con las ventas de canciones en un momento dado. También se crean listas de éxitos respecto a estilos musicales.

### PLAYLISTS

Las playlists o listas de reproducción, son una de las funcionalidades más populares en los servicios de streaming. Un usuario crea una lista con un título y añade una serie de canciones. Conforme se van añadiendo canciones a dicha playlist, el servicio va sugiriendo canciones que podrían estar relacionadas (ya sea por similitud entre canciones o porque se han encontrado en otras playlists con títulos parecidos). En este caso los usuarios pueden compartir sus listas con otros, aparte de que el servicio también crea múltiples playlist que pueden seguirse. En la Figura 8 mostramos una captura de una playlist creada en Spotify.



Figura 8. Ejemplo de una playlist creada por Spotify



## 2.4. PROYECTOS SIMILARES

Para finalizar este capítulo, vamos a hacer un repaso de algunos proyectos similares al que se va a tratar en este trabajo, disponibles en Internet. Antes de comenzar, cabe destacar que los proyectos que hemos localizado sólo generan una playlist a partir de un cantante o canciones indicadas (salvo en un caso que es capaz de crear listas mediante estado anímico y género). Otra consideración a tener en cuenta es que todos los proyectos son para la creación de listas en *Spotify*.

### 2.4.1. MAGICPLAYLIST

Este proyecto ofrece la posibilidad de crear una playlist a partir de una canción indicada o bien mediante el estado anímico junto a un género musical. Una vez que la lista de reproducción ha sido creada, nos ofrece la oportunidad de guardarla en *Spotify* [12].

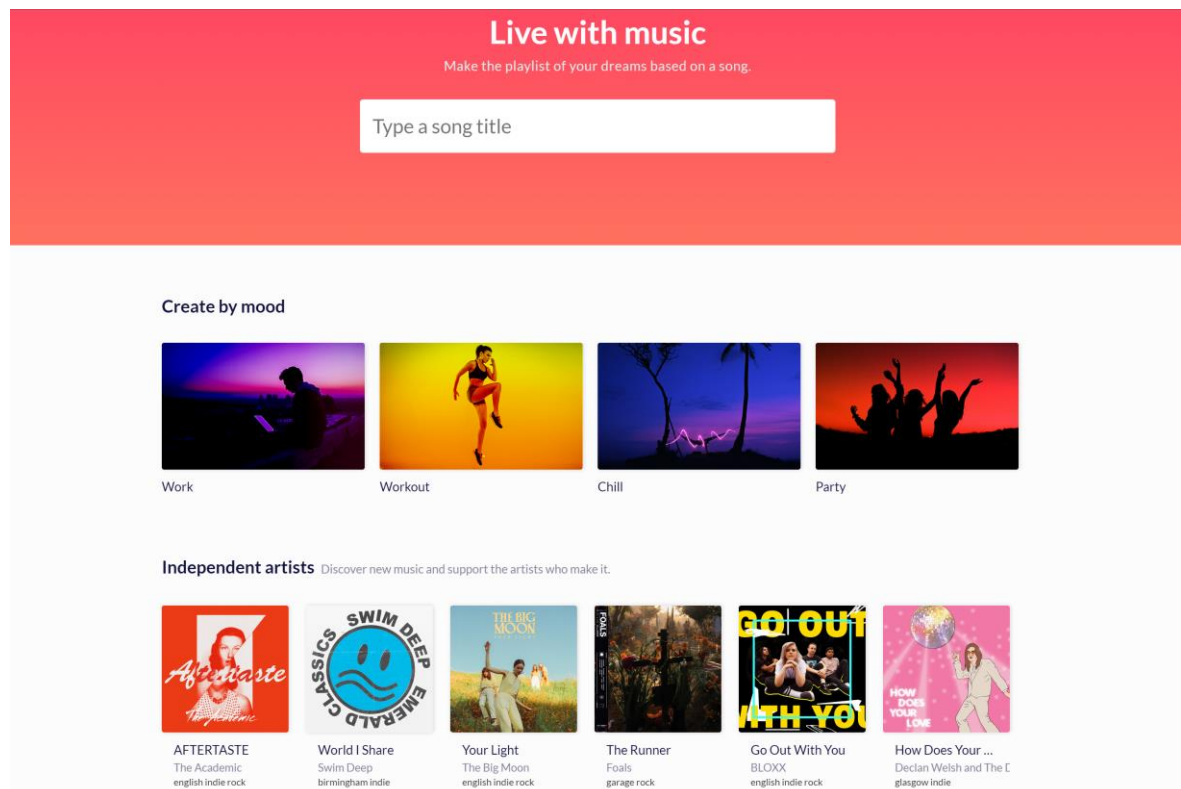


Figura 9. Captura de la página web del proyecto MagicPlaylist

### 2.4.2. PLAYLIST GENERATOR

Con este generador, a partir de una canción inicial y una final elegidas por un usuario, junto al número de canciones que se desea, se genera una playlist. La forma en la que se genera es la siguiente: se crea un grafo donde los nodos son las canciones incluidas en el conjunto de datos del proyecto y se intenta encontrar un camino entre las canciones que hemos indicado. En este caso también podemos importarla a *Spotify* [13].

#### Generate Your Playlist

To generate your playlist, you must select the first and last song of the playlist and set the desired number of songs.

First Song

Artist: Taylor Swift

Select Artist

Song: Shake It Off

Select Song

Last Song

Artist: Katy Perry

Select Artist

Song: Hot N Cold

Select Song

Number of songs in the playlist:

15

Create Playlist

Generated Playlist:

1. Shake It Off by Taylor Swift
2. This Is How We Do by Katy Perry
3. Chasing The Sun by The Wanted
4. Animals by Maroon 5
5. Beauty And A Beat by Justin Bieber
6. Want U Back by Cher Lloyd
7. Rude Boy by Rihanna
8. Party In The U.S.A. by Miley Cyrus
9. Sweet Dreams by Beyoncé
10. Hot N Cold by Katy Perry

*Figura 10. Ejemplo de una recomendación con Playlist Generator*

### 2.4.3. SPOTIFY PLAYLIST GENERATOR

En este caso, a partir de un cuadro de diálogo indicamos qué tipo de playlist queremos (recomendaciones por artistas, álbumes o canciones) de forma programática. Un ejemplo es el siguiente: Si en el cuadro de dialogo introducimos el fragmento de código que mostramos a continuación:

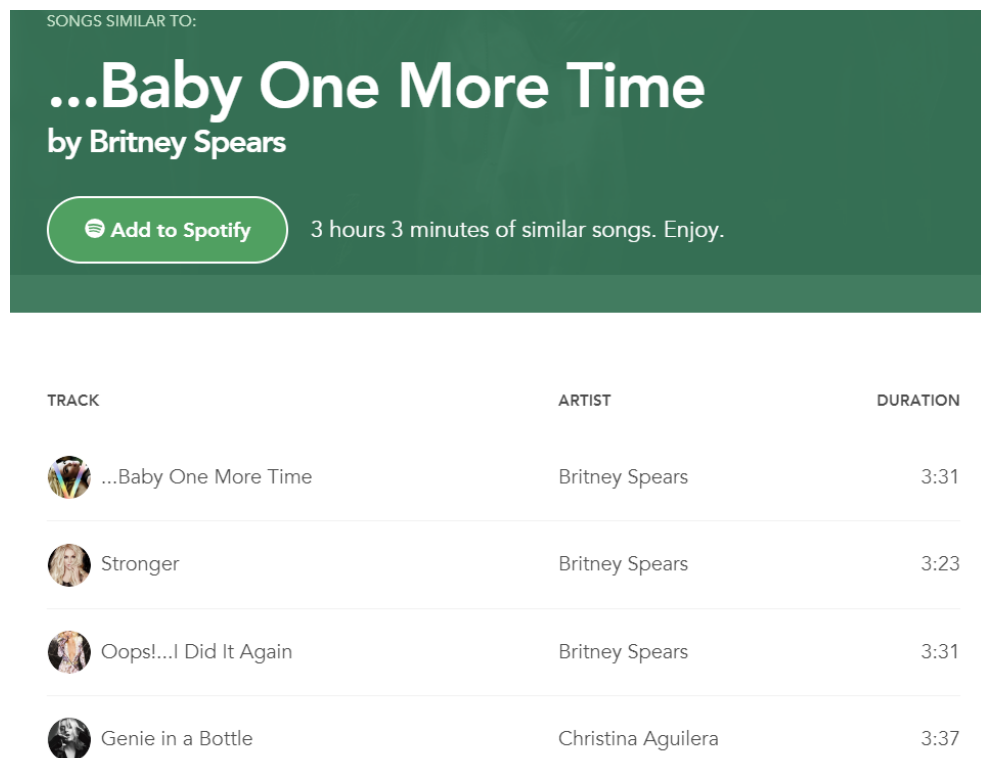
```
## All Deerhunter tracks, ordered by Last.fm rating

#order by lastfm
#artist Deerhunter
```





crearía una playlist titulada “All Deerhunter tracks, ordered by Last.fm rating” y con el top de canciones del artista *Deerhunter* ordenadas por la puntuación que el servicio *Last.fm* ofrece [14].

### 2.4.4. SPOTALIKE

Para este último proyecto, en fase beta, aparte de usar las herramientas ofrecidas por *Spotify* también hace uso de las disponibles por el portal de música *Last.fm*. Su formato es similar al de un buscador: se introduce el nombre de una canción y el sistema muestra los resultados (canciones en nuestro caso) que ha encontrado [15]. En la Figura 11 se muestra un ejemplo de ejecución.



The screenshot displays the 'SONGS SIMILAR TO:' section for the song '...Baby One More Time' by Britney Spears. It features a green header with the song title and artist, a button to 'Add to Spotify', and a duration of '3 hours 3 minutes of similar songs. Enjoy.' Below this is a table of recommended songs.

TRACK	ARTIST	DURATION
 ...Baby One More Time	Britney Spears	3:31
 Stronger	Britney Spears	3:23
 Oops!...I Did It Again	Britney Spears	3:31
 Genie in a Bottle	Christina Aguilera	3:37

*Figura 11. Ejemplo de uso del proyecto Spotalike*



# CAPÍTULO 3. METODOLOGÍA Y DESARROLLO

## 3.1. METODOLOGÍA

Para el desarrollo de este Trabajo de Fin de Grado, vamos a emplear la metodología *CRISP-DM* (*Cross Industry Standard Process for Data Mining*), la cual nos provee de una descripción normalizada del ciclo de vida de un proyecto de minería de datos.

El modelo *CRISP-DM* se desarrolla en 6 pasos, los cuales explicamos a continuación y podemos ver de forma gráfica en la Figura 12:

1. Comprensión del negocio: Nos centramos en comprender los objetivos y requisitos del proyecto desde una perspectiva comercial, para luego convertir este conocimiento en una definición de problema de minería de datos y un plan preliminar [16] [17].

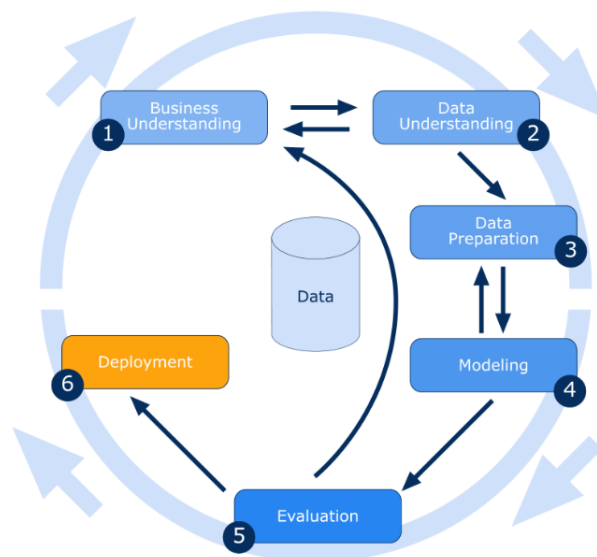


Figura 12. Fases del modelo de referencia *CRISP-DM* [39]

2. Estudio y comprensión de los datos: Comenzamos con una recopilación de datos inicial y continuamos con las actividades para comprenderlos, identificar problemas de calidad, descubrir las primeras ideas o detectar subconjuntos interesantes para formar hipótesis de información oculta [18].
3. Preparación de los datos: La fase de preparación cubre todas las actividades para construir el conjunto final de datos a partir de los datos iniciales que se encuentran sin procesar [18].
4. Modelado: Se seleccionan y aplican técnicas de modelado. Dado que algunas técnicas, como las redes neuronales, tienen requisitos específicos con respecto a la forma de los datos, podemos encontrarnos el caso de tener que volver a la fase de preparación de datos [18].
5. Evaluación: Una vez que se han construido uno o varios modelos que parecen tener una alta calidad en base a las funciones de pérdida que se hayan seleccionado, estos deben ser probados para garantizar que se generalicen bien al usar datos nuevos, no usados en el entrenamiento y que todos los problemas comerciales clave se hayan considerado suficientemente. El resultado final es la selección de los modelos con mejores resultados [18].
6. Despliegue: En general, consiste en desplegar una representación de código del modelo en un sistema operativo para calificar o categorizar nuevos datos invisibles, a medida que surjan, y crear un mecanismo para el uso de esa nueva información en la solución del problema comercial inicial. Es importante destacar que la representación del código también debe incluir todos los pasos de preparación de datos que conducen al modelado, para que el modelo trate los nuevos datos sin procesar de la misma manera que durante el desarrollo del modelo [17].

## 3.2. DESARROLLO

En esta sección vamos a hacer un breve repaso de todas las tecnologías que hemos empleado para el trabajo, así como de los medios hardware donde se ha realizado y probado el sistema.

### 3.2.1. LENGUAJE DE PROGRAMACIÓN

En nuestro caso, hemos empleado el lenguaje de programación *Python* (en su versión 3.6). *Python* resulta muy útil en el desarrollo de aplicaciones complejas tanto numéricas como científicas. Debido a su versatilidad, *Python* es uno de los lenguajes más empleados en la actualidad. Un ejemplo de ello es que, durante 2019, se ha convertido en el segundo lenguaje más usado en *GitHub* superando así al lenguaje *Java*.

Para el campo de ciencia de datos, *Python* se ha convertido en un lenguaje muy popular gracias a sus principales paquetes empleados para tal fin. Algunos ejemplos son *Pandas*, *NumPy* y *SciPy*, que producen excelentes resultados para trabajos de análisis de datos. Si nos encontramos con la necesidad de emplear gráficos, *matplotlib* es una gran opción para ello, al igual que *scikit-learn* para tareas de *machine learning*.

### 3.2.2. HERRAMIENTAS DE DESARROLLO

Para la creación del conjunto de datos, la creación del modelo, pruebas, análisis, etc., se han empleado los siguientes entornos:

- Jupyter Notebook: Entorno de programación interactivo, mediante el cual podemos introducir texto en formato *markdown* junto a código *Python* haciendo uso de los *notebooks* o libretas.
- Visual Studio Code: Conocido editor de código de *Microsoft*, el cual hemos empleado a la hora de visualizar los archivos que hemos creado para almacenar los

datos, tanto en formato *CSV* como *JSON*, y para el desarrollo de scripts empleados para ejecutar los procesos relacionados con la recopilación de datos y el entrenamiento del modelo.

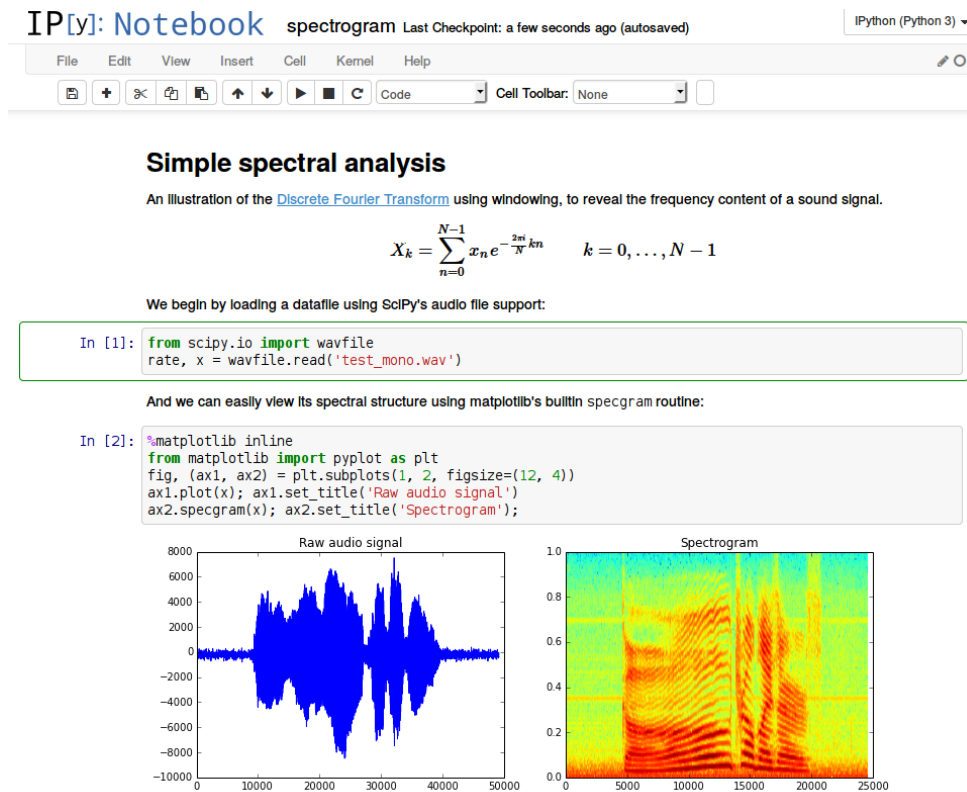


Figura 13. Libreta de ejemplo ejecutada en Jupyter Notebook

### 3.2.3. BIBLIOTECAS

Aunque hemos empleado numerosas bibliotecas o paquetes a la hora de realizar las tareas de desarrollo, vamos a nombrar algunas de las más destacadas:

- Spotipy: Biblioteca de *Python* para usar la *WebAPI* de *Spotify*.
- Numpy: Ofrece mayor soporte a vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con estos elementos.



- Pandas: Es una biblioteca de código abierto para *Python* que proporciona estructuras de datos y herramientas de análisis de alto rendimiento y fáciles de emplear.
- NLTK: Este *toolkit* es una de las bibliotecas más potentes para procesamiento del lenguaje.
- SciPy: Biblioteca de código abierto basada en *Python*, que se utiliza en matemáticas, ciencia e ingeniería.
- Scikit-Learn: Biblioteca empleada para problemas de aprendizaje automático o *machine learning*.
- LightFM: Es una biblioteca que contiene numerosos algoritmos utilizados en los sistemas de recomendación.

### 3.2.4. HARDWARE EMPLEADO

#### EQUIPO PRINCIPAL

Surface Book 2	
<b>Procesador</b>	Intel Core i7-8650U CPU @ 1.90GHz, 2112 Mhz, 4 procesadores principales, 8 procesadores lógicos
<b>Memoria RAM</b>	16 GB
<b>Almacenamiento</b>	512 GB
<b>Sistema operativo</b>	<ul style="list-style-type: none"> <li>• Windows 10 (versión 1909)</li> <li>• Ubuntu 18.04 LTS (bajo <i>Windows Subsystem for Linux</i>)</li> </ul>

Tabla 1. Especificaciones técnicas del equipo principal

### CAPÍTULO 3. Metodología y desarrollo

Adicionalmente al equipo principal, ha sido necesario alquilar una serie de máquinas virtuales a un proveedor de *cloud* para agilizar algunos procesos en el desarrollo de este TFG. En este caso se ha empleado *Microsoft Azure* utilizando la suscripción para estudiantes [19].

A continuación, mostramos el tipo de máquinas virtuales empleadas junto a las especificaciones y el número de instancias utilizadas:

Máquina virtual – Serie B (Uso general)	
<b>Tamaño</b>	Standard_B2ms
<b>CPUs virtuales</b>	2
<b>Memoria RAM</b>	8 GB
<b>Almacenamiento</b>	16 GB
<b>Instancias</b>	4

*Tabla 2. Especificaciones técnicas de máquina virtual Serie B*

Máquina virtual – Serie Fsv2 (Proceso optimizado)	
<b>Tamaño</b>	Standard_B2ms
<b>CPUs virtuales</b>	16
<b>Memoria RAM</b>	32 GB
<b>Almacenamiento</b>	128 GB
<b>Instancias</b>	1

*Tabla 3. Especificaciones técnicas de máquina virtual 'Serie Fsv2'*

## CAPÍTULO 4. CONJUNTO DE DATOS

---

En este capítulo vamos a hablar de cómo se ha construido el conjunto de datos que utilizaremos en este trabajo. Inicialmente se intentó buscar un conjunto que contuviera playlists de *Spotify*, pero no se encontró ninguno disponible a excepción del *Million Playlist Dataset (MPD)* [4]. Este conjunto se utilizó en la competición de ciencia de datos *RecSys Challenge* [3], en el año de 2018, y sólo fue proporcionado a los participantes que se habían registrado en ella. Se intentó contactar con los organizadores de la competición para que nos facilitasen el *dataset*, pero tras no obtener respuesta se decidió crear un dataset basado en la información proporcionada por la competición y en un fragmento publicado por uno de los grupos participantes [20].

### 4.1. MILLION PLAYLIST DATASET

El *MPD* contiene un millón de listas de reproducción generadas por usuarios de *Spotify*. Estas listas se crearon durante el periodo comprendido entre enero de 2010 y octubre de 2017. Cada playlist contiene su título, la lista de canciones (incluyendo metadatos de cada una), información de edición (número de veces que se ha editado y la fecha de la última edición) y otros datos diversos sobre la playlist [20].

Este conjunto de datos está dividido en 1.000 ficheros, donde cada fichero contiene 1.000 listas de reproducción. Los ficheros están nombrados de la siguiente forma:

`mpd.slice.STARTING_PLAYLIST_ID_-_ENDING_PLAYLIST_ID.json`

Por ejemplo, las primeras 1.000 listas del *MPD* están contenidas en un fichero llamado *mpd.slice.0-999.json*, mientras que las últimas 1.000 lo están en el fichero llamado *mpd.slice.999000-999999.json* [20].

## CAPÍTULO 4. Conjunto de datos

Cada bloque de listas se corresponde a un diccionario *JSON* con los siguientes campos: *info* y *playlists*.

### CAMPO INFO

El campo *info* es un diccionario que contiene información general sobre el bloque en particular [20]:

- slice: Porción de listas que contiene.
- version: Versión del conjunto de datos.
- generated\_on: Fecha en la que ha sido generado.

```
{'generated_on': '2017-12-03 08:41:42.057563',  
  'slice': '0-999',  
  'version': 'v1'}
```

Figura 14. Ejemplo del campo “info” de un archivo *JSON* del MPD

### CAMPO PLAYLIST

Este campo es un array que contiene una lista de 1.000 diccionarios. Cada diccionario, que equivale a una playlist, contiene los siguientes campos [20]:

- pid: Identificador numérico de la playlist dentro del conjunto.
- name: Título de la lista.
- modified\_at: Fecha de modificación.
- num\_artists: Número total de artistas únicos que contiene la playlist.
- num\_albums: Número total de álbumes únicos que contiene la playlist.
- num\_tracks: Número total de pistas que contiene la playlist.
- num\_followers: Número total de seguidores que tiene la playlist.
- num\_edits: Número de veces que se ha editado una playlists. Se considera que las pistas agregadas en un intervalo de 2 horas han sido agregadas en una sola sesión de edición.
- collaborative: Si la playlist puede modificarse por cualquier usuario de *Spotify*.

- duration\_ms: Duración total en milisegundos de la playlist.
- tracks: Lista de diccionarios con información sobre las pistas. Cada diccionario, que equivale a una pista, contiene los siguientes campos:
  - *track\_name*: Título de la pista.
  - *track\_uri*: Dirección *Spotify* de la pista.
  - *album\_name*: Título del álbum.
  - *album\_uri*: Dirección *Spotify* del álbum.
  - *artist\_name*: Nombre del artista.
  - *artist\_uri*: Dirección *Spotify*.
  - *duration\_ms*: Duración en milisegundos de la pista.
  - *pos*: Posición de la pista dentro de la playlist.

En la Figura 15 se muestra un ejemplo de un ítem del diccionario perteneciente al campo *playlist*.

#### 4.1.1. CRITERIOS DE CONSTRUCCIÓN

El *Million Playlist Dataset* se creó muestreando listas de reproducción de los miles de millones que existen en *Spotify* y que los usuarios han creado a lo largo de los años. Las listas de reproducción que cumplen los siguientes criterios, se seleccionan al azar [20]:

- Creada por un usuario que reside en los Estados Unidos y es mayor de 13 años.
- A la hora en la que fue generado el *MPD*, la lista era pública.
- Contiene entre 5 y 250 pistas.
- Contiene al menos 3 artistas diferentes.
- Contiene al menos 2 álbumes diferentes.
- No contiene pistas locales (pistas que el usuario tiene en su dispositivo).
- Tienen al menos un seguidor (sin incluir al propietario de la playlist).
- Se han creado después del 1 de junio de 2010 y antes del 1 de diciembre de 2017.
- No tiene un título ofensivo.
- No tiene un título orientado a adultos si la playlist fue creada por un usuario menor de 18 años.

## CAPÍTULO 4. Conjunto de datos

Criterios como el país de residencia del usuario, fecha de creación o edad del usuario, no son posibles de obtener en nuestro caso, puesto que la *WebAPI* de *Spotify* no proporciona dicha información.

```
{
  "name": "musical",
  "collaborative": "false",
  "pid": 5,
  "modified_at": 1493424000,
  "num_albums": 7,
  "num_tracks": 12,
  "num_followers": 1,
  "num_edits": 2,
  "duration_ms": 2657366,
  "num_artists": 6,
  "tracks": [
    {
      "pos": 0,
      "artist_name": "Degiheugi",
      "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
      "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
      "track_name": "Finalement",
      "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az406UML",
      "duration_ms": 166264,
      "album_name": "Dancing Chords and Fireflies"
    },
    // 10 tracks omitted
    {
      "pos": 11,
      "artist_name": "Mo' Horizons",
      "track_uri": "spotify:track:7iwx00eBzeSSSy6xfESyWN",
      "artist_uri": "spotify:artist:3tuX54dqqS8LsGUvNzgrpP",
      "track_name": "Fever 99\u00b0",
      "album_uri": "spotify:album:2Fg1t2tyOSGWkVYH1FfXVf",
      "duration_ms": 364320,
      "album_name": "Come Touch The Sun"
    }
  ],
}
```

*Figura 15. Ejemplo de una playlist contenida en el MPD*

### 4.1.2. OBSERVACIONES

Como mencionamos al principio de este capítulo, obtuvimos un fragmento del *MPD* que consta de 4.000 listas. Tras realizar un análisis exploratorio mediante una libreta *Jupyter*, hemos observado y establecido otros criterios que pueden resultarnos de utilidad:

- El título de la playlist tiene entre 2 y 50 caracteres, sin contar los espacios.
- El número máximo de palabras que contiene un título es de 10.
- Los títulos sólo contienen caracteres del alfabeto latino y emoticonos.
- Si el título contiene texto, podrán aparecer hasta un máximo de 10 emoticonos.
- Si el título sólo está compuesto por emoticonos, habrá un máximo de 6.

Otra observación que podemos destacar, y que nos servirá de ayuda a la hora de buscar playlists, es que gran parte de los títulos están relacionados con alguna de las siguientes temáticas:

- Géneros musicales
- Estaciones del año
- Estados de ánimo
- Actividades

En la libreta *Jupyter* titulada “*Parte 0 - Estudio del Million Playlist Dataset (MPD)*”, que podemos encontrar en el soporte adjunto, se realiza un estudio detallado del fragmento del conjunto que hemos obtenido.

## 4.2. BÚSQUEDA DE PLAYLISTS

Una vez que hemos establecido los criterios que vamos a tomar para elegir las playlists, pasamos a la fase de búsqueda. Los resultados obtenidos contienen la información básica de las listas de reproducción, como por ejemplo el identificador de la playlist, el título o número de canciones que contiene, ya que en el proceso de búsqueda no devuelve la

## CAPÍTULO 4. Conjunto de datos

información completa de cada una de ellas (lista de pistas que conforman la playlist, seguidores, ...).

Para realizar las búsqueda hemos creado un fichero de texto plano con las 3.000 palabras más frecuentes usadas en inglés, obtenidas de *EF Education First* [21], y también hemos recopilado un conjunto de géneros musicales (pop, rock, jazz, ...), épocas (80's, 90's, ...), periodos festivos (*Christmas, Halloween...*) y otras palabras relacionadas con actividades (*running, relax, ...*) y estados de ánimo (*happy, angry, ...*) que han sido repartidas en los 4 ficheros siguientes:

- `split_lists/splitlist-ages.txt`
- `split_lists/splitlist-genres.txt`
- `split_lists/splitlist-special1.txt`
- `split_lists/splitlist-special2.txt`

Con la idea de trabajar de una forma más cómoda con la *WebAPI* de *Spotify*, hemos hecho uso de *Spotipy*. Esta es la biblioteca de *Python* con la cual obtenemos acceso completo a todos los *endpoints* y el soporte a la autorización por parte del usuario (para algunas características se requiere que el usuario confirme que desea otorgar el acceso).

Como ayuda para seguir el proceso de búsqueda, ya que conlleva varias horas para completarse, hemos creado una función que se encarga de enviar mensajes mediante *Telegram* empleando un *bot* para poder informarnos sobre la evolución del proceso y que también nos avisa en caso de que exista algún error. También hemos almacenado un registro del proceso de búsqueda en un fichero de texto.

### 4.2.1. PROCESO DE BÚSQUEDA

Antes de iniciar el proceso de búsqueda de las playlists, hemos realizado los siguientes pasos:



1. Lectura del fichero que contiene la lista de términos: `files/list-words_eng.txt`
2. Clasificación de las palabras según su letra inicial en un diccionario cuya clave es la letra y su valor es la lista de palabras.
3. Una vez que hemos clasificado las palabras, procedemos a volcarlas en ficheros independientes de la siguiente manera:

`split_lists/splitlist-{initial_letter}.txt`

El directorio donde almacenamos estas listas es el mismo donde se encuentran ubicadas las listas con los términos especiales que hemos mencionado anteriormente, ya que el proceso de búsqueda cargará los datos de esa misma carpeta.

El motivo por el cual se han decidido almacenar las palabras en varios ficheros es para que el proceso de búsqueda pueda hacerse de forma escalonada y/o distribuida (varias máquinas y/o procesos).

Una vez creados los ficheros, comenzamos con el proceso de búsqueda mediante la función ***process\_split\_lists***. Su funcionamiento es el siguiente:

1. Obtenemos una lista con los ficheros que contienen los términos que hemos clasificado.
2. Para cada archivo, cargamos la lista de palabras contenida y creamos una copia que utilizaremos para consultar los términos que hay pendientes de buscar.
3. Establecemos el fichero, en formato *CSV*, donde se almacenarán los resultados obtenidos para el conjunto de palabras correspondiente. Dicho fichero tendrá la siguiente nomenclatura: `sets/set-{set_id}.csv`
4. Para cada término, hacemos una consulta a *Spotify* mediante la función ***playlist\_search***, con la cual obtenemos una lista de diccionarios con los resultados de la búsqueda. Cada diccionario contiene los siguientes campos:

## CAPÍTULO 4. Conjunto de datos

identificador, número de pistas y nombre de la playlist. Posteriormente guardamos los resultados en el fichero *CSV*, eliminamos la palabra de la lista de pendientes y actualizamos el fichero, para que en caso de existir algún problema el proceso se reanude por el último termino en el que ha quedado pendiente de buscar.

Tanto en la función encargada de procesar las listas de palabras como en la función que se encarga de realizar las búsquedas en *Spotify* se han introducido varios retardos, puesto que los términos de uso de la *API* requieren que no se realicen llamadas de forma masiva para evitar el mal funcionamiento del servicio.

Para ejecutar la búsqueda, hemos empleado 4 máquinas virtuales de serie *B* (ver Tabla 2) en *Microsoft Azure*. Cada máquina ha ejecutado simultáneamente 3 procesos de búsqueda.

Una vez completado el proceso de búsqueda, se han obtenido los resultados mostrados en la Tabla 4.

<b>Número de términos buscados</b>	3.154
<b>Playlists obtenidas tras la búsqueda de términos</b>	15.797.992
<b>Tiempo empleado (aproximación)</b>	16,5 horas
<b>Espacio ocupado en memoria</b>	715 MB

*Tabla 4. Resultados del proceso de búsqueda*

También se ha creado una copia de seguridad de todos los ficheros que han sido generados por el proceso de búsqueda en un archivo comprimido.

En la libreta Jupyter titulada “*Parte 1 - Búsqueda de playlists*”, puede verse con más detalle el funcionamiento del proceso de búsqueda.

### 4.3. PREPARACIÓN DEL PROCESO DE DESCARGA

Tras realizar el proceso de búsqueda, hemos obtenido un número elevado de resultados. Ante tal cantidad, hemos realizado un pre-filtrado de playlists para evitar la descarga de listas innecesarias. Este proceso, que realizaremos empleando *pandas*, consiste en lo siguiente:

- Eliminación de playlists repetidas mediante el identificador que le asigna *Spotify*.
- Filtrado por número de pistas, para quedarnos con aquellas que contengan entre 5 y 250 pistas.
- Filtrado por título, eliminando aquellas con 1 carácter o con más de 50 y las que contienen más de 9 palabras.

Hemos repetido estos criterios tras la descarga de playlists, ya que en el caso del número de pistas tenemos que ver si alguna de ellas son pistas que el usuario tiene en su colección personal. El filtrado por títulos nos interesa aplicarlo con más detalle en un número reducido de listas de reproducción. Otro motivo por el que se ha realizado es para evitar que los resultados de búsqueda no concuerden con los de descarga, si algunos usuarios modifican las playlists que hemos encontrado durante el proceso de búsqueda y de descarga.

Los resultados que hemos obtenido tras eliminar aquellas playlists que no nos resultan de interés se muestran en la Tabla 5.

Por último, hemos agrupado los identificadores de las listas de reproducción que deseamos descargar en conjuntos de 1.000 y los hemos almacenado en ficheros para proceder a la descarga. Al igual que en el proceso de búsqueda, hemos creado varios ficheros para ejecutar el proceso en varias máquinas. También hemos guardado una copia de seguridad de estos ficheros en un archivo comprimido.

En la libreta de *Jupyter* titulada “*Parte 2 - Preparación para la descarga de playlists*” podemos ver todo el proceso de descarga realizado, así como los resultados obtenidos.

<b>Playlists obtenidas en el proceso de búsqueda</b>	15.797.992
<b>Playlist repetidas</b>	3.714.899
<b>Playlist con más de 250 pistas o menos de 5</b>	1.533.999
<b>Playlist con títulos no válidos</b>	154.475
<b>Playlist restantes tras el filtrado</b>	10.394.619

*Tabla 5. Resultados del proceso de preparación de descarga*

#### 4.4. DESCARGA DE PLAYLISTS

Tras la preparación de datos, hemos seleccionado un conjunto de identificadores de playlists para descargar. Este conjunto es bastante grande, puesto que al no disponer de información suficiente con los resultados que hemos obtenido en el proceso de búsqueda (identificador, título y número de pistas que contiene la lista de reproducción), necesitamos descargarlos para decidir cuáles de ellos seleccionamos para nuestro conjunto de datos.

Para facilitar la tarea de descarga, hemos establecido 3 de los criterios anteriores para decidir si descargamos o no la lista de reproducción:

- La playlist debe tener al menos 1 seguidor.
- En el momento de descarga, la playlist debe ser pública.
- La playlist no contiene pistas locales (pertenecen al usuario y no están en *Spotify*).

Gracias a esto hemos reducido el tiempo de descarga, puesto que, como primero hacemos una llamada a la *API* para ver si la playlist es pública y tiene más de 1 seguidor, la respuesta es inmediata (ya que el sistema no tiene que devolver la lista completa). Otra ventaja es la del espacio de almacenamiento, ya que no almacenamos las playlists que no cumplen los criterios anteriores.

Al igual que hicimos en el proceso de búsqueda, hemos empleado el *bot* de *Telegram* para seguir la evolución del proceso de descarga y si existe algún problema poder solucionarlo lo antes posible. También hemos almacenado un registro del proceso de descarga en un fichero de texto.

Para la descarga de playlists, hemos creado una serie de métodos con la finalidad de hacer más sencillo el proceso de búsqueda:

- ***get\_playlist\_tracks***: Si una playlist contiene más de 100 pistas, *Spotify* no devuelve en una misma llamada el contenido completo de la lista. Con esta función, cuando deseemos descargar la lista de pistas, basta con indicar el usuario (opcional) y el identificador de la lista de reproducción para la cual queremos obtener el listado completo.
- ***clean\_tracks\_info***: No toda la información que nos devuelve *Spotify* sobre las pistas que conforman una playlist nos es útil. Emplearemos esta función para recopilar los datos que nos interesan y obtener las pistas en una lista de diccionarios.
- ***pl\_has\_local\_files***: Mediante esta función comprobamos de manera sencilla si una playlist contiene pistas locales, pertenecientes al usuario, o no.
- ***supdate\_to\_timestamp***: Cuando la llamada a la *WebAPI* nos devuelve una playlist, para cada pista podemos ver en qué fecha se agregó a la lista. Dicha fecha está en un formato que no es estándar. Esta función la emplearemos para resolver dicho problema convirtiendo el formato de fecha de *Spotify* a *TimeStamp*.
- ***pl\_has\_incomplete\_content***: Otra comprobación que debemos realizar es si, para alguna pista de la playlist, el contenido de *artist\_name*, *album\_name* o *track\_name* está vacío. Existen casos en el que la pista que hay agregada a una playlist ya no está disponible.

- ***get\_spotify\_error***: En determinadas ocasiones, *Spotify* nos devuelve una excepción cuando no ha podido acceder a la lista. Principalmente, nos encontramos con dichas excepciones cuando la lista no está disponible o bien porque el usuario la ha eliminado en el tiempo transcurrido entre el proceso de búsqueda y el de descarga. Con esta función cogemos la información de la excepción que nos da *Spotify* y la convertimos en un formato más fácil de entender.
- ***pl\_check\_requirements***: Mediante esta función comprobamos si una lista es publica y tiene más de un seguidor, para descargarla en caso de que lo cumpla. Estas restricciones se comprueban de forma separada a la de pistas locales, puesto que si no se cumplen las dos primeras no es necesario comprobar la última.

### 4.4.1. PROCESO DE DESCARGA

El proceso que realizamos para descargar las playlist es muy similar al de búsqueda: leemos el contenido de los ficheros de la carpeta que contienen los identificadores de las playlists a descargar, los procesamos y para cada uno realizamos la descarga de la información desde *Spotify*.

A continuación, detallamos cómo funciona el método *download\_playlist*, que mediante un identificador y los campos que nos interesa extraer de la playlist, genera un diccionario con el contenido de ésta:

1. Mediante la biblioteca *Spotipy*, descargamos el contenido de la playlist.
2. Extraemos el número de seguidores, ya que no nos interesa la lista de usuarios que siguen la playlist.
3. Obtenemos la lista completa de pistas que conforman la playlist mediante la función que hemos creado. Como vimos anteriormente, si la lista tiene más de 100 elementos sólo se descargaban los 100 primeros.

4. Comprobamos que la lista no contiene pistas locales pertenecientes al usuario.
  - a. En caso de que no tenga pistas locales:
    - i. Extraemos la información más relevante de las pistas.
    - ii. Calculamos el número total de pistas y lo almacenamos con la clave *num\_tracks* dentro del diccionario que vamos a devolver.
  - b. En caso de que existan pistas locales, vaciamos el diccionario a devolver.
5. Devolvemos el diccionario que contiene la playlist que hemos indicado.

Una vez explicado cómo nos encargamos de descargar el contenido de una playlist, pasamos a ver cómo se procesan los ficheros que contienen los identificadores de aquellas listas que nos interesa descargar. Para cada fichero realizamos lo siguiente:

1. Leemos los identificadores que tiene el fichero que estamos procesando.
2. Creamos una lista donde almacenaremos aquellos identificadores que se han omitido (ya sea porque la lista es privada, no tiene seguidores o no esté disponible).
3. Para cada identificador que hayamos obtenido de la lista:
  - a. Comprobamos que cumpla los requisitos que hemos establecido.
    - i. En caso de que los cumpla, comprobamos que no tiene pistas locales y lo almacenamos en un fichero *JSON* dentro una carpeta temporal, donde se almacenan de forma individual.
    - ii. En caso de que tenga pistas locales, la omitimos y agregamos el identificador a la lista de playlist descartadas.
  - b. Si no cumple los requisitos establecidos, descartamos el identificador, lo añadimos a la lista de descartados y continuamos.
4. En caso de que se produzca una excepción en el proceso de descarga de playlists, esperaremos 5 minutos y volvemos a intentarlo.

5. Una vez que se han descargado todas las playlist del fichero:
  - a. Leemos las playlist de los ficheros *JSON* que hemos almacenado en la carpeta temporal (comprobando que no sean aquellas que hemos descartado) y los almacenamos en una lista llamada *pls\_list*.
  - b. Una vez que se han leído todas las playlist, guardamos la lista que las contiene en un único fichero *JSON* dentro de la carpeta que hemos indicado y lo comprimimos.
6. Borramos el fichero que contiene los identificadores de las playlist que hemos descargado y los ficheros con las playlists de forma individual que están almacenados en la carpeta temporal.

Al igual que hicimos en el proceso de búsqueda, introducimos una serie de retardos para evitar el mal funcionamiento del servicio y cumplir con los términos de uso de *Spotify*.

Para la descarga de playlists, hemos usado las 4 máquinas virtuales que tenemos creadas en *Microsoft Azure* y ejecutamos 3 procesos de descarga en cada una.

Una vez terminado el proceso de descarga de playlists, hemos obtenido los resultados mostrados en la Tabla 6.

<b>Playlists descargadas</b>	3.122.640
<b>Tiempo empleado</b>	152 horas ( $\approx$ 6'3 días)
<b>Espacio ocupado en memoria</b>	124 GB

*Tabla 6. Resultados del proceso de descarga*

El proceso de descarga completo se encuentra disponible en la libreta *Jupyter* titulada “*Parte 3 - Descarga de playlists*”.



## 4.5. FILTRADO DE PLAYLISTS

Una vez que hemos descargado aquellas listas de reproducción que nos interesaba estudiar con más detalle, hemos realizado el proceso completo de filtrado para quedarnos con 1.000.000 de playlists. También hemos aprovechado este filtrado para elegir un determinado número de las listas sobrantes para posteriormente construir un conjunto de prueba.

Durante este proceso hemos repetido los siguientes filtros, que ya habíamos aplicado durante el proceso de búsqueda y el de descarga:

- Tienen menos de 5 pistas o más de 250 pistas.
- El título contiene menos de 2 caracteres o más de 50 (sin contar espacios).
- El título contiene más de 9 palabras.

Como hemos comentado previamente, durante el proceso de búsqueda y el proceso de descarga han transcurrido varios días y el usuario podría haber modificado el título o las pistas que pertenecen a la playlist.

También hemos decidido aplicar los siguientes criterios:

- Playlists que contengan artistas poco frecuentes (sólo aparecen en 1 o 2 listas).
- Número de seguidores inferior a 2.

En el caso de artistas poco frecuentes, realizamos este paso para que el conjunto de pistas de nuestro *dataset* no sea muy elevado. Aumentar el número mínimo de seguidores de 1 a 2 se debe a que consideramos que aquellas playlists que sólo sigue 1 persona no son relevantes.

Previamente hemos extraído la información de las playlists descargadas, salvo sus listas de pistas, a un fichero *CSV* para cargarlo a un *dataframe* de *pandas* y aplicar el filtrado sobre él. Este proceso se ha realizado con la función ***get\_downloaded\_playlists\_df***.

## CAPÍTULO 4. Conjunto de datos

Como no hemos extraído la lista de pistas que pertenecen a cada playlist, hemos guardado en el *dataframe* el nombre del fichero comprimido donde se ubica el contenido completo de la playlist para que, cuando nos sea necesario hacer uso de él, podamos obtenerla de forma rápida.

Para crear dicho *dataframe*, hemos definido varias nuevas funciones. Algunas de ellas resultan necesarias ya que, por ejemplo, el número único de artistas y álbumes lo tenemos que calcular con la lista de pistas de cada playlist, al igual que necesitamos otra función para calcular las veces que se ha editado una playlist, el cual obtendremos gracias a las fechas en las que las pistas de la lista de reproducción fueron agregadas. Las funciones más destacadas son las siguientes:

- ***get\_playlist\_edits***: Haciendo uso de la fecha en la que ha sido agregada una pista a la playlist, calculamos el número de veces que ha sido editada. Recordemos que las pistas agregadas en una ventana de 2 horas corresponden a una sesión de edición. En nuestro caso, hemos contado también como una edición la primera vez que se añadieron pistas a la playlist. Así pues:
  - Si el número de ediciones es 1, la playlist no se ha modificado tras su creación.
  - Si el número de ediciones es igual o superior a 2, la playlist se ha editado al menos una vez tras su creación.
- ***get\_playlist\_info***: Con esta función obtenemos el número único de artistas y de pistas, empleando los conjuntos de *Python*, junto a la duración total de la lista sumando el tiempo de cada pista.

Una vez que el *dataframe* con la información de las playlists ha sido creado, comprobamos si existen variables con valores perdidos. Tras la consulta, hemos obtenido los resultados mostrados en la Tabla 7.

Como podemos observar, existen valores perdidos para las variables *num\_followers*, *modified\_at* y *num\_edits*. Para resolver estas situaciones hacemos lo siguiente:

- En el caso del número de seguidores, consideramos que un valor *NaN* equivale a que la lista de reproducción no tiene ningún seguidor. Por lo tanto, la descartamos.
- Para las dos variables restantes: si no existe fecha de última modificación ni número de veces que se ha editado una playlist, la descartamos por no poder calcular dichos valores.

Una vez que hemos tratado los valores perdidos, el *dataframe* ya está listo para poder realizar el filtrado de playlists. En la Figura 16 mostramos un fragmento del *dataframe* con el que hemos trabajado.

Variable	Número de valores perdidos
<b>collaborative</b>	0
<b>name</b>	0
<b>num_tracks</b>	0
<b>num_followers</b>	255
<b>num_artists</b>	0
<b>num_albums</b>	0
<b>duration_ms</b>	0
<b>modified_at</b>	724
<b>num_edits</b>	724
<b>file_name</b>	0

*Tabla 7. Valores perdidos para el dataframe de playlists descargadas*

id	collaborative	name	num_tracks	num_followers	num_artists	num_albums	duration_ms	modified_at	num_edits
9hUw5qK0K2GDwH	False	Love Letter	17	65.0	16	16	3601989	1.556283e+09	5.0
Wt8mB0CayapCcRr	False	skin deep	35	1.0	32	35	7146193	1.558387e+09	16.0
QuXhb3YqO24HHw	False	junior year	250	1.0	113	184	57370653	1.556406e+09	113.0
oQ33CvNMiZ1uH	False	BOOM BOOM	20	1.0	16	20	4120665	1.555591e+09	2.0
RPsHlbXILXlhYoxF6	False	Born in the Wrong Era	168	1.0	112	146	36941201	1.559002e+09	86.0

Figura 16. Fragmento del dataframe empleado para el filtrado de playlists

### 4.5.1. PROCESO DE FILTRADO DE PLAYLISTS

El proceso de filtrado que hemos aplicado sobre el *dataframe* de *pandas* ha sido realizado en 4 bloques:

1. Filtrado por número de pistas, artistas y álbumes.
2. Filtrado de playlists con idénticas características.
3. Filtrado por número de ediciones.
4. Filtrado por títulos.

### FILTRADO POR NÚMERO DE PISTAS, ARTISTAS Y ÁLBUMES

Recordemos que en el caso del número de pistas, artistas y álbumes deben de cumplirse los siguientes criterios:

1. El número de pistas de una playlist estará comprendido entre 5 y 250.
2. La playlist tendrá, al menos, 3 artistas diferentes.
3. La playlist tendrá, al menos, 2 álbumes diferentes.

## FILTRADO DE PLAYLISTS CON IDÉNTICAS CARACTERÍSTICAS

Hay casos en los que existen listas de reproducción iguales, pero con diferente título y pertenecientes a usuarios distintos. Para dichos casos, vamos a quedarnos con el primer resultado y borrar el resto. Consideramos que las playlists son idénticas si los valores para las siguientes variables son iguales:

- Duración de la playlist.
- Número de canciones.
- Número de artistas diferentes.
- Numero de álbumes diferentes.

Puedan darse casos extremos en los que, coincidiendo esos datos, pueda tratarse de otra lista de reproducción distinta.

## FILTRADO POR NÚMERO DE EDICIONES

Este filtrado se realiza exactamente igual que el que hemos hecho con los álbumes, artistas y pistas. Basta con buscar aquellas filas cuya variable *num\_edits* sea inferior a 2 y eliminarlas.

## FILTRADO POR TÍTULOS

Este último grupo de filtros es el más largo y complejo de todos los que hemos aplicado. Como recordamos, pudimos filtrar por títulos tras la búsqueda de playlists. El motivo por el que no se ha hecho así es que el proceso resulta menos costoso con 3 millones de playlists que con 10 millones, por lo que hemos tenido que adoptar por esta estrategia.

Los últimos grupos de filtros que vamos a aplicar para los títulos de las playlists son los siguientes:

## CAPÍTULO 4. Conjunto de datos

1. Tiene entre 5 y 50 caracteres (eliminando espacios en blanco).
2. Tienen menos de 10 palabras.
3. En caso de que el título contenga emoticonos:
  - a. Si el título también contiene texto, el número máximo de emoticonos es 10.
  - b. Si el título sólo contiene emoticonos, el número máximo será de 4.
4. Los caracteres del título pertenecen al alfabeto latino, al conjunto de caracteres comunes, o son emoticonos.
5. Filtrado de títulos por idioma.
6. Filtrado de títulos ofensivos.

Como podemos observar, hemos realizado un filtrado por idioma. Puesto que unos de los requisitos originales del *MPD* era que las playlists debían ser de usuarios residentes en Estados Unidos, nosotros vamos a quedarnos con aquellas playlists cuyo idioma sea el inglés.

### 1. FILTRADO POR NÚMERO DE CARACTERES

Para filtrar por el número de caracteres, hemos considerado las siguientes dos fases para aplicarlo al *dataframe* de *pandas* que estamos empleando:

- a. Guardamos los identificadores de aquellas playlists que tienen menos de 5 caracteres y en la otra guardamos los que tengan más de 10 caracteres (salvo el espacio).
- b. Una vez que tenemos las listas con los identificadores, las empleamos para borrar las correspondientes filas del *dataframe*.

### 2. FILTRADO POR NÚMERO DE PALABRAS

Para aplicar este filtro, procedemos de forma similar al filtrado por número de caracteres. En particular, obtenemos los identificadores que, tras crear una lista con las palabras separarlas por un espacio, contienen un número de palabras superior a 10, y eliminamos las filas correspondientes del *dataframe*.

## FILTRADO POR EXISTENCIA DE EMOTICONOS

Para filtrar los títulos de las playlist mediante la existencia de emoticonos, hemos usado la biblioteca *emoji* que hay disponible para *Python* junto a una serie de funciones que hemos creado para facilitar la tarea.

La función que más relevancia tiene es *invalid\_name\_emojis*, la cual nos ayuda a identificar aquellos títulos de playlists que no cumplen con lo establecido:

- Poseen más de 10 emoticonos si el título no contiene texto.
- Poseen más de 4 emoticonos si el título contiene texto.

## 3. FILTRADO POR TIPO DE CARACTERES

Como comentamos en el estudio del fragmento del *MPD*, los caracteres pertenecían al alfabeto latino, al conjunto de caracteres comunes y/o eran emoticonos. En dicho caso, lo comprobamos mediante expresiones regulares.

En esta ocasión, para acotar más el número de playlists y controlar mejor aquellos caracteres con los que vamos a trabajar, hemos limitado el alfabeto a los caracteres que aparecen a continuación:

```
' . , : ; @ # $ % & ~ ° | ) ( > < } { ] [ + - * / % ? ¡ ! _ -- 0 1 2 3 4 5 6 7 8 9
a b c d e f g h i j k l m n o p q r s t u v w x y z ñ á é í ó ú à è ì ò ù ä ë ï ö ü â ê î ô û
```

*Figura 17. Alfabeto de caracteres válidos para los títulos de las playlists*

Hemos ignorado el carácter “\” debido a que la barra invertida se usa a menudo para secuencias de escape (“\n”, “\t”, ...).

## CAPÍTULO 4. Conjunto de datos

Una vez que hemos establecido el alfabeto, creamos una función llamada ***contains\_invalid\_chars*** con la cual comprobamos que un título contiene caracteres válidos y emoticonos. Esta función devuelve un valor booleano: si alguno de los caracteres que contiene el título no son válidos obtenemos *True*, en caso contrario la función devuelve *False*.

Por último, mediante una función de *pandas* eliminamos aquellas filas en las que la columna *name* del *dataframe* se obtiene el valor *True* una vez aplicada la función anterior.

### 4. FILTRADO POR IDIOMA

Para filtrar los títulos mediante el idioma empleado, hemos utilizado el servicio *Text Analytics* que se encuentra en los *Cognitive Services* de *Microsoft Azure* [22]. Para realizar este proceso, hemos usado como guía el ejemplo publicado en la documentación oficial.

Para emplear *Text Analytics*, hemos obtenido una clave de acceso tras crear el servicio en nuestra cuenta de *Azure*. Para obtenerla, hemos seguido la documentación proporcionada por el servicio [23].

Una vez que hemos configurado el servicio, se ha creado una función para hacer uso de él. Esta función la hemos llamado ***get\_names\_language***. Mediante ella, obtenemos el idioma de cada uno de los títulos de las playlist que tenemos en nuestro *dataframe*. Posteriormente hemos guardado los resultados en un fichero *JSON* comprimido. También hemos guardado una copia de dicha información por los siguientes motivos:

- El proceso puede durar más de 1 hora.
- Si volvemos a realizar las llamadas al servicio, se nos vuelve a facturar por el uso que hacemos (en caso de usar alguna de las instancias de pago). En caso de hacer uso de la instancia gratuita, podríamos agotar innecesariamente las 5.000 instancias al mes que nos facilitan.



Una vez que hemos implementado el proceso de detección del idioma y obtenido la identificación para cada título, construimos un *dataframe* con el identificador de la playlist, el título y el resultado que nos ha dado *Text Analytics*. Tras este paso, procedemos del siguiente modo:

1. Algunos de los títulos que hemos obtenido se han identificado como *Unknown*. Podría darse el caso de que fueran playlists que sólo contuvieran iconos. Procedemos a identificarlas y en caso de que sean de dicho tipo, sustituimos la identificación por *Emoji* y establecemos una puntuación de 1.
2. Eliminamos aquellas listas en las que el idioma predominante no es el inglés (etiqueta '*en*') y aquellas que se han identificado como *Unknown* o *Emoji*.
3. Por último, en los casos donde los títulos se han identificado en idioma inglés, pero su puntuación es inferior a 1, nos quedamos únicamente con aquellos que tienen una puntuación igual o superior a 0.75.

## 5. FILTRADO POR TÍTULOS CON CONTENIDO OFENSIVO

Como tras el filtrado anterior el idioma predominante es el inglés, hemos obtenido un listado de palabras que podrían ser ofensivas de la web *Free Web Headers* [24], para eliminar aquellas playlist que las contengan. En la web podemos encontrar varios listados de diferentes proveedores. En nuestro caso hemos elegido las palabras que *Google* tiene almacenadas en una 'lista negra' para revisar los comentarios de *YouTube* donde se encuentre alguna de ellas.

### 4.5.2. RESULTADOS

Tras aplicar el proceso de filtrado a las playlists que hemos obtenido durante el proceso de descarga, hemos obtenido los resultados mostrados en la Tabla 8.

RESULTADOS DEL PROCESO DE FILTRADO	
Playlists disponibles antes de aplicar los filtros	3.122.640
Playlists cuyo número de pistas no es valido	8.051
Playlists cuyo número de artistas y álbumes no es valido	159.638
Playlists repetidas	3.772
Playlists editadas menos de 2 veces	183.152
Playlists eliminadas aplicando los filtros para títulos	375.618
Playlists con un número de seguidores inferior a 2	1.117.885
Playlists que contienen artistas poco frecuentes	255.176
Playlists disponibles tras aplicar los filtros	2.392.399

*Tabla 8. Resultados obtenidos tras el proceso de filtrado*

Tras todo este proceso empleado para el filtrado, hemos almacenado en un archivo *CSV* para utilizarlo en el proceso de generación del conjunto de datos.

En la libreta Jupyter titulada “*Parte 4 - Filtrado de playlists*” que podemos encontrar en el soporte adjunto, podemos ver todo el proceso de filtrado de forma más detallada.

## 4.6. GENERACIÓN DEL CONJUNTO DE DATOS

Tras habernos quedado con un número de playlists cercano a 1.000.000, procedemos a crear nuestro conjunto de datos. Dicho conjunto está compuesto por dos fragmentos: el primero de ellos contiene el millón de playlists que vamos a usar para entrenar el modelo y el segundo está compuesto por 10.000 playlists incompletas, que emplearemos como conjunto de prueba.

### 4.6.1. CONJUNTO DE ENTRENAMIENTO

En primer lugar, vamos a cargar el archivo *CSV* que obtuvimos tras finalizar el proceso de filtrado a un *dataframe*, descartando las listas que van desde la posición 1.000.000 en adelante. Las playlists que se encuentran a partir de dicha posición son las que utilizaremos para crear el conjunto de prueba.

Como hemos visto al inicio del capítulo, el *MPD* está compuesto por 1.000 ficheros donde cada fichero contiene 1.000 playlists. En nuestro caso lo hemos dividido del mismo modo.

Las playlists han sido almacenadas en los ficheros siguiendo el orden en el que aparecen en el *dataframe*. Para saber a qué bloque/fichero pertenece cada playlist, lo hemos hecho empleando la siguiente fórmula:

$$\text{bloque}_n = \text{techo} \left( \frac{\text{playlists}_{\text{posicion}}}{\text{num\_playlists}_{\text{fichero}}} \right)$$

A continuación, hemos creado un diccionario donde almacenamos a qué bloque pertenece cada conjunto. Este diccionario tendrá como clave el id de la playlist y su valor será el bloque donde será almacenada. Por ejemplo, si tomamos como referencia la lista que aparece en la posición 652.386 y cuyo id es *0GtTxbghpueJjaRxS6GOqp*, al consultar el diccionario vemos que pertenece al bloque 652.

Antes de comenzar a generar nuestro *dataset*, nos falta reunir la información completa de cada playlist. Las playlists completas se encuentran dispersas entre más de 10.000 ficheros *JSON*, puesto que cuando las descargamos las guardamos con respecto al fichero de identificadores que las contenía. Para obtener dicha información, hemos creado un diccionario cuya clave es el nombre del fichero *JSON* y su valor es la lista de playlist que necesitamos extraer.

Una vez se ha leído el contenido de un fichero y extraído aquellas playlists que nos interesaban, realizamos un volcado de listas de reproducción a nuevos archivos. Por ejemplo,

## CAPÍTULO 4. Conjunto de datos

si realizamos 5 volcados de datos pertenecientes al bloque *598*, tendremos 5 ficheros diferentes. A estos volcados de datos, aparte de indicarse a qué bloque pertenecen, se les añadirá un sufijo “*.dump\_NUMERO-VOLCADO*” para identificarlo.

Tras concluir el proceso de agrupación de las playlists que pertenecen a un bloque, nos encargaremos de unir en un sólo fichero todos aquellos volcados que pertenezcan a un bloque. Este método funciona de la siguiente manera:

1. Omitiendo los sufijos “*.dump\_*”, identificamos a qué bloques pertenecen los distintos volcados.
2. Mediante los nombres que hemos obtenido al eliminar los sufijos, procedemos a identificar los volcados que pertenecen a los diferentes bloques que tenemos.
3. Leemos todos los volcados pertenecientes a un bloque.
4. Unimos los volcados en un único fichero.
5. Eliminamos los volcados tras crear el fichero único.

Por ejemplo, los siguientes ficheros pertenecen al bloque *109*:

- *track-lists-set\_109.dump0.json*
- *track-lists-set\_109.dump1.json*
- *track-lists-set\_109.dump2.json*
- *track-lists-set\_109.dump3.json*
- *track-lists-set\_109.dump4.json*

pertenecen al bloque *109*. Tras leerlos, generamos un nuevo fichero y borramos los anteriores (que contenían la terminación “*.dump\_*”):

*track-lists-set\_109.json*

Como ya tenemos el contenido de las playlists agrupado por bloque, procedemos a crear el conjunto de datos que emplearemos para el entrenamiento del modelo. El proceso que vamos a seguir, para cada bloque/fichero que debemos crear, es el siguiente:

1. Establecemos el valor 'info' del fichero JSON (cabecera con los datos pertenecientes al bloque).
2. Leemos el fichero que contiene las listas de pistas de las playlist pertenecientes al bloque.
3. Para cada id de la lista de reproducción que pertenece al bloque, cargamos sus datos junto a la lista de pistas y los guardamos en el valor 'playlist' del fichero JSON a crear.
4. Generamos el fichero para posteriormente comprimirlo y quedarnos únicamente con el fichero .zip.

En la libreta “*Parte 5 - Generación del DataSet*” está disponible todo el proceso de generación para este fragmento del conjunto.

### 4.6.2. CONJUNTO DE PRUEBA

Para crear el conjunto de prueba, nos basaremos también en el conjunto que se ofreció como reto en el *RecSys Challenge 2018* [20].

Dicho conjunto se utilizaba para evaluar el sistema de recomendación creado. Los participantes del reto tenían el conjunto de forma incompleta y, una vez creado el sistema, enviaban los resultados a la organización para devolverles la puntuación obtenida.

En nuestro caso, vamos a crear dos ficheros: uno que contendrá las playlists con su contenido completo y otro con las playlists incompletas (con la posibilidad de que el nombre de la playlist no figure).

## CAPÍTULO 4. Conjunto de datos

El conjunto consta de un único diccionario almacenado en un fichero *JSON*, con los siguientes campos [20]:

- date: Fecha en la que se ha generado.
- version: Versión del conjunto de datos.
- playlists: Array de 10.000 playlist incompletas. Cada elemento de la lista es un diccionario que posee los siguientes campos:
  - *pid*: Identificador de la playlist.
  - *name*: Título de la lista. (Opcional)
  - *num\_holdouts*: Número de pistas que han sido omitidas de la lista.
  - *tracks*: Lista de diccionarios con información sobre las pistas (la lista puede estar vacía).
  - *num\_samples*: Número de pistas que contiene la lista. Donde:

$$num\_samples == len(tracks)$$

- *num\_tracks*: Número total de pistas que contiene la lista. Donde:

$$num\_tracks = num\_samples + num\_holdouts$$

## CATEGORÍAS

Para el conjunto de prueba hemos establecido 10 categorías, con 1.000 playlists en cada una, correspondientes a los retos empleados en el *RecSys Challenge 2018* [20] [25]. Los títulos de las categorías o retos que se establecido para probar el modelo, son los siguientes:

1. Predicción de pistas para una playlist dado sólo su título.
2. Predicción de pistas para una playlist dado su título y la primera pista.
3. Predicción de pistas para una playlist dado su título y las primeras 5 pistas.
4. Predicción de pistas para una playlist dadas las primeras 5 pistas (sin título).
5. Predicción de pistas para una playlist dado su título y las primeras 10 pistas.
6. Predicción de pistas para una playlist dadas las primeras 10 pistas (sin título).

7. Predicción de pistas para una playlist dado su título y las primeras 25 pistas.
8. Predicción de pistas para una playlist dado su título y 25 pistas aleatorias.
9. Predicción de pistas para una playlist dado su título y las primeras 100 pistas.
10. Predicción de pistas para una playlist dado su título y 100 pistas aleatorias.

### RESTRICCIONES

A la hora de construir el conjunto que utilizaremos para la fase de prueba, hemos aplicado las siguientes restricciones a las playlists:

- Todas las pistas del conjunto de prueba están en el conjunto final (MPD).
- Todas las pistas por predecir (holdout) están en el conjunto final.

### CREACIÓN DEL CONJUNTO DE PRUEBA

Para la lectura de las playlists con su información completa, hemos empleado el mismo procedimiento que en la creación del fragmento que empleamos para el entrenamiento. En este caso, vamos a crear 2 ficheros: el primero de ellos contendrá el contenido completo de todas las playlists (que utilizaremos para comprobar los resultados de la predicción) y otro con las playlists incompletas, que emplearemos para realizar las predicciones, atendiéndonos a la categoría que pertenecen.

Una vez que hemos leído las playlists que se encuentran a partir de la posición 1.000.000 del *dataframe* obtenido tras la lectura de los resultados del proceso de filtrado, comprobamos que no tenga ninguna pista que no esté en el fragmento que empleamos para el entrenamiento.

Tras eliminar aquellas listas que no podemos emplear en el conjunto de prueba, procedemos a elegir qué listas pueden formar parte de una de las categorías en las que se ha dividido el conjunto. Lo único que debemos tener en cuenta es que las playlists tengan más pistas que el número de elementos que se ofrecen en cada categoría. Por ejemplo: Para la

## CAPÍTULO 4. Conjunto de datos

categoría "Predecir las pistas para una playlist dado su título y 100 pistas aleatorias", tenemos que asegurarnos de que las listas que vamos a elegir tengan más de 100 pistas.

Puesto que ya hemos establecido qué playlists conforman cada categoría, vamos a leer los datos de cada una e incorporarlos a una lista. Con este proceso crearemos el conjunto de prueba completo (figuran todos los nombres y pistas de las playlists) y lo guardaremos en un fichero comprimido.

El último paso es eliminar aquellas pistas las playlists de tal manera que se cumpla lo establecido en la categoría. Por ejemplo: para la categoría “Predicción de pistas para una playlist dado su título y 25 pistas aleatorias”, cuando eliminemos las pistas de las playlists que conforman dicha categoría únicamente tendremos el título y 25 pistas seleccionadas al azar.

Llegado a este punto, generamos el archivo que contiene las playlists de forma incompleta y lo almacenamos en un fichero comprimido.

El proceso de creación del conjunto que utilizaremos para la fase de prueba del modelo, podemos encontrarlo en la libreta *Jupyter* titulada “*Parte 6 - Generación del DataSet de pruebas*”.



## CAPÍTULO 5. CONSTRUCCIÓN DEL MODELO DE RECOMENDACIÓN

---

Una vez que hemos creado nuestro conjunto de datos, pasamos a definir un modelo con el que realizar la predicción de pistas para una playlists. A lo largo de este capítulo, vamos a hablar de la biblioteca que hemos empleado y el motivo de su elección, los pasos que hemos seguido para realizar el preprocesamiento y explicaremos el proceso de entrenamiento.

### 5.1. LIGHTFM

*LightFM* es una implementación en *Python* de una serie de algoritmos de recomendación para la retroalimentación implícita y explícita, incluida la implementación eficiente de las funciones de pérdida *BPR* (*Bayesian Personalised Ranking*) y *WARP* (*Weighted Approximate-Rank Pairwise*). Es fácil de usar, rápido (a través de la estimación de modelos multiproceso) y produce resultados de alta calidad [26].

También permite incorporar metadatos de elementos y usuarios en los algoritmos tradicionales de factorización matricial, representando a cada usuario y elemento como la suma de las representaciones latentes de sus características, permitiendo así que las recomendaciones se generalicen a nuevos elementos (a través de las características del elemento) y a nuevos usuarios (a través de las características del usuario) [26].

El motivo de la elección de *LightFM* se debe a que se basa en un modelo híbrido que supera en desempeño a los modelos colaborativos y los basados en contenido, consiguiendo solucionar el problema del *arranque en frío* o aquellos escenarios de escasa interacción,

siendo su rendimiento muy similar al de los modelos de factorización de matrices, donde los datos de interacción son abundantes [27].

En la libreta *Jupyter* titulada “*Parte 7 – Introducción a LightFM*”, se muestra un pequeño tutorial de cómo se crea un modelo, se realiza la fase de preprocesamiento, se calculan las métricas, etc.

### 5.1.1. MODELO LIGHTFM

Como hemos comentado, *LightFM* es un modelo híbrido de recomendación. Dicho modelo aprende representaciones latentes en un espacio de alta dimensión para usuarios y elementos de manera que codifica las preferencias del usuario sobre los elementos [26].

Las representaciones de usuario y elemento son expresadas en función de sus características. Por ejemplo, si la película *El caballero oscuro* se describe mediante las siguientes características: *acción*, *superhéroes* y *comics*, su representación estará formada por la suma de las representaciones latentes de estas características. Las incrustaciones se aprenden a través de métodos de *descenso estocástico del gradiente* [26].

Algunas de las funciones de pérdida que están presentes en *LightFM* son las siguientes:

- *BPR (Bayesian Personalised Ranking)*: Maximiza la diferencia de predicción entre un ejemplo positivo y un ejemplo negativo elegido al azar. Es útil cuando sólo hay interacciones positivas y se desea optimizar el área bajo la curva ROC (*AUC*) [26].
- *WARP (Weighted Approximate-Rank Pairwise)*: Maximiza el rango de ejemplos positivos al muestrear repetidamente ejemplos negativos hasta que se encuentra el rango que lo incumpla. Es útil cuando solo hay interacciones positivas y se desea optimizar la parte superior de la lista de recomendaciones (*precision@k*) [26].

En *LightFM* también nos encontramos con dos algoritmos de optimización para gradiente descendiente: *Adagrad* y *Adadelta*.

## 5.2. PREPROCESAMIENTO

En este apartado, vamos a hablar del proceso que hemos empleado para preparar los datos y así puedan ser usados en el modelo *LightFM*. Hemos dividido el preprocesamiento en dos partes:

1. Parte 1: Conversión a formato *CSV* y creación de características de playlist.
2. Parte 2: Creación de matrices de expansión.

### 5.2.1. CONVERSIÓN DE DATOS A FORMATO CSV

Para crear los ficheros que contienen el *dataset* con el conjunto de playlists, empleamos el formato *JSON* y lo almacenamos en archivos comprimidos debido a su elevado tamaño: 7,5 GB (43,3 GB sin comprimir). También nos encontramos con el caso de que tenemos la información repetida de las pistas, puesto que cada playlist contiene toda la información completa de éstas.

Para solucionar estos problemas y poder trabajar mejor con los datos, vamos a convertirlos al formato *CSV*, con lo que tendremos la información repartida en varias tablas y evitaremos tener información repetida de forma innecesaria. Se ha creado un fichero/tabla para cada tipo de ítem del conjunto:

- Álbumes.
- Artistas.
- Pistas.
- Información sobre las playlists.

## CAPÍTULO 5. Construcción del modelo de recomendación

- Información sobre las playlists del conjunto de prueba.
- Lista de pistas de las playlists (para ambos conjuntos)

Adicionalmente, hemos creado una tabla que contiene la lista de artistas de las playlists, junto al número de veces que aparece cada artista en ellas.

La forma en que hemos procedido para convertir el conjunto de datos al formato *CSV* es la siguiente:

1. Leemos los ficheros comprimidos que contienen el archivo *JSON*. Para cada playlist que se encuentra dentro del archivo, hacemos lo siguiente:
  - a. Extraemos la lista de pistas y almacenamos el resto de la información en una fila del fichero *CSV* correspondiente.
  - b. Leemos la lista de pistas de la playlist y por cada pista añadimos una fila en el fichero *CSV*, que empleamos para almacenar las pistas que conforman una playlist con el número de playlists, la posición que ocupa y el identificador *Spotify* de la pista.
  - c. Conforme vamos leyendo las listas de pistas, añadimos a un diccionario un elemento cuya clave es el identificador *Spotify* de la pista y su valor es un nuevo diccionario con la información de la pista. De esta forma obtendremos todas las pistas que se encuentran en nuestro conjunto.
2. Una vez hemos leído todos los ficheros, volcamos la información del diccionario donde hemos almacenado las pistas a un archivo *CSV*.
3. Para el fichero que contiene la información sobre las pistas, extraemos la información de los artistas y álbumes a dos nuevos ficheros, dejando sólo en el fichero original los identificadores *Spotify* del artista y del álbum para cada pista.

Para el conjunto de playlists incompletas realizamos un proceso similar al anterior, salvo que no extraemos la información de las pistas (todas las pistas contenidas en este conjunto ya se encuentran en el conjunto que acabamos de procesar), y la información básica de la playlist la almacenamos en otro fichero *CSV*. La lista de pistas de cada playlist, las almacenamos en el mismo archivo donde se encuentran las del conjunto de entrenamiento.

Como los identificadores *Spotify* contienen unos “prefijos” que resultan únicos en el caso de los álbumes, artistas y pistas, los eliminamos (ya que en caso de ser necesario podemos volver a añadirlos). Ejemplo:

*spotify:album:0MLT0iC5ZYKFGeZ8h3D4rd* → *0MLT0iC5ZYKFGeZ8h3D4rd*

Para que las tablas sean más fáciles de entender y ocupen menos tamaño, hemos establecido nuestro propio identificador, al que llamaremos *PID*, para álbumes, pistas y artistas. De esta manera las relaciones entre los elementos de las tablas, que hemos almacenado en ficheros *CSV*, se hacen empleando estos identificadores, y no por el código alfanumérico de *Spotify*. Por ejemplo, donde antes nos encontrábamos con:

```
track_pid,track_name,duration_ms,artist_id,album_id,track_id
0,No Control,237306,21KpdYIquYJUEiEcPO2cnI,5F9BLbIkEvApFC8fLNVKYR,3G0MLTNw2AX3Xdbhrj330S
```

tras realizar este cambio, obtenemos el siguiente resultado:

```
track_pid,track_name,duration_ms,artist_pid,album_pid,track_id
0,No Control,237306,0,0,3G0MLTNw2AX3Xdbhrj330S
```

Llegados a este punto, ya disponemos del conjunto de datos convertido del formato *JSON* a *CSV*.

En la Figura 18, Figura 19 y Figura 20 podemos ver algunos ejemplos de algunas de las tablas contenidas en los ficheros *CSV*, empleando un *dataframe* de *pandas* para mostrar su contenido.

## CAPÍTULO 5. Construcción del modelo de recomendación

	name	collaborative	modified_at	num_albums	num_tracks	num_followers	num_edits	duration_ms	num_artists
pl_pid									
0	Low viscosity vibes	False	1559722465	38	58	3	14	15191903	25
1	dalanda 🍷	False	1558851313	100	104	4	75	22361466	92
2	freeze pops	False	1552263755	25	52	2	7	10643802	13
3	Golden Oldies	False	1556948665	90	98	2	34	25418594	64

Figura 18. Muestra del dataframe que contiene la información de las playlists

pos	track_pid	pos	track_pid	pos	track_pid	pos	track_pid
pl_pid		pl_pid		pl_pid		pl_pid	
0	0	0	5	0	10	0	20
0	1	0	6	0	11	0	21
0	2	0	7	0	12	0	22
0	3	0	8	0	13	0	23
0	4	0	9	0	14	0	24

Figura 19. Muestra del dataframe que contiene las pistas que pertenecen a una playlist

	track_name	track_id	duration_ms	artist_pid	album_pid
track_pid					
0	No Control	3G0MITNw2AX3Xdbhrj33OS	237306	0	0
1	City Lights	3R9H16eUSv5vJ9DEgMG2Lu	369693	0	0
2	Wake Up the Neighborhood	.60pgLDeu9thOdkPhLZHPcQ	218040	0	212099
3	Runnin' With the Devil - (Remastered 2015) [Re...	0xAoC8qDVcl7xAQEe0PTUo	214960	173	212099

Figura 20. Muestra del dataframe que contiene la información de las pistas

### 5.2.2. CREACIÓN DE CARACTERÍSTICAS DEL USUARIO

A continuación, vamos a convertir los títulos en etiquetas, en las que cada una corresponderá a una característica.

Como podemos recordar, algunas playlists contienen emoticonos. Para este caso creamos un diccionario con el cual cambiar dichos elementos por palabras. El proceso que vamos a seguir para crear el diccionario es el siguiente:

1. Obtenemos aquellas playlists que contienen emoticonos y las almacenamos en una lista.
2. Extraemos la colección de emoticonos que aparecen en todas las playlists.
3. Para cada emoticono, obtenemos las playlists donde aparece y para cada una de ellas limpiamos el texto eliminando los signos de puntuación, de tal forma que nos quedamos con letras, números y emoticonos.
4. Una vez que hemos eliminado los signos de puntuación, para cada lista eliminamos las palabras *vacías* (aquellas que no aportan significado) y realizamos el proceso de lematización para cada palabra (extraemos el lema correspondiente).
5. Una vez que las listas donde figura el emoticono no contienen signos de puntuación ni palabras *vacías*, y hemos extraído el lema de todas las palabras, contamos el número de ocurrencias de las palabras y nos quedamos con las N palabras que más aparecen para ese emoticono.
6. Una vez se han procesado todos los emoticonos y se han obtenido las N palabras que aparecen con más frecuencia para cada emoticono, almacenamos los resultados en un diccionario y hacemos una copia de seguridad volcando los datos en un fichero *JSON*.

En la Figura 21 podemos observar una muestra del contenido del diccionario que empleamos para convertir emoticonos en palabras.

```
'💡': ['study', 'light', 'lit', 'yellow', 'lost']
'👑': ['king', 'queen', 'girl', 'disney', 'rap']
'🗣️': ['talk', 'yell', 'bake', 'sing', 'belt']
'🔒': ['special', 'one', 'free', 'taz', 'hardcastle']
'🍳': ['breakfast', 'bacon', 'wake', 'bake', 'vibe']
'🎉': ['party', 'student', 'weekend', 'birthday', 'wine']
'💛': ['yellow', 'happy', 'love', 'country', 'summer']
'📄': ['office']
'🍺': ['cool', 'beer']
'🥗': ['salad', 'yo', 'mama']
'🌟': ['jewish', 'rock', 'real']
```

Figura 21. Fragmento el diccionario de emoticonos

Una vez creado el diccionario de emoticonos, comenzamos con el proceso de creación de etiquetas (que en nuestro caso serán las características de las playlists). El proceso que realizamos para cada título de las playlists es el siguiente:

1. Extraemos los emoticonos a una nueva lista.
2. Eliminamos signos de puntuación, los emoticonos y los espacios en blanco que aparecen dos o más veces seguidas (dejando solamente un espacio).
3. Mediante el nombre de la playlists ‘limpio’ y la lista de iconos que aparecían en ella (en caso de haberlos), generamos las etiquetas:
  - a. En el caso de las palabras, aplicamos *stemming* a cada una (extraemos la raíz de la palabra) y las incorporamos a un conjunto.
  - b. Para los emoticonos, sustituimos cada uno por las etiquetas que obtenemos tras consultar el diccionario y los incorporamos al conjunto donde se encuentran las palabras.
4. Por último, agrupamos las etiquetas (palabras que contiene el conjunto) en un string en el que cada una se encuentra separada por “|”. Por ejemplo, si tenemos un conjunto (tag1,tag2,tag3) y unimos sus términos en un *string* mediante la barra vertical, obtenemos: “tag1|tag2|tag3”

Con estos datos, creamos un nuevo dataframe en el que almacenaremos el identificador de la playlist, el título de ésta y las etiquetas que hemos extraído, y lo exportamos a un archivo *CSV*. En la siguiente figura, podemos los primeros elementos del dataframe que hemos creado.

	name	tags
pl_pid		
0	Low viscosity vibes	low viscos vibe
1	dalanda 🐉	imagine dragon game dalanda
2	freeze pops	pop freez
3	Golden Oldies	oldi golden

Figura 22. Muestra del dataframe que contiene las etiquetas de los nombres de playlists



### 5.2.3. CREACIÓN DE MATRICES DE EXPANSIÓN

*LightFM* requiere que los datos que se le pasen al modelo sean matrices de expansión. En el caso de las características de usuario serán *matrices CSR* (matrices de expansión comprimidas) [28], mientras que las interacciones entre usuarios e ítems serán *matrices COO* (matriz de expansión en formato de coordenadas) [30]. En caso de que las interacciones entre usuario e ítem contengan un peso, también se requerirá que estas sean una matriz *COO*.

Para facilitarnos esta tarea, en la biblioteca de *LightFM* podemos encontrar la clase ***lightfm.data.Dataset***, la cual proporciona una serie de funciones para construir las matrices de interacción y las de características.

Las funciones que empleamos a la hora de crear las matrices de expansión con la biblioteca *LightFM*, son las siguientes:

- ***build\_interactions***: Crea la matriz de interacción. En caso de que en nuestro conjunto de datos las interacciones tengan pesos, obtendremos dos matrices. La primera de ellas es la matriz de interacciones, compuesta por unos (en caso de que el usuario haya interactuado) y ceros (en caso de que no haya interacción). La segunda matriz devolverá lo mismo, salvo que en las posiciones donde había un uno encontraremos el peso de la interacción [26]. En la Figura 23 y Figura 24 podemos ver un ejemplo de matriz de interacción y una matriz con los pesos de las interacciones.
- ***build\_item\_features***: Crea la matriz que contiene las características de los ítems. Sus columnas corresponderán a las características, las filas corresponden a los ítems y los datos son 0 y 1 (1 en caso de que el ítem tenga esa característica).
- ***build\_user\_features***: Similar al caso anterior. Crea la matriz que contiene las características de los usuarios. Sus columnas se corresponden con las características, las filas se corresponden con los usuarios y los datos son 0 y 1 (1 en caso de que el usuario tenga esa característica).

## CAPÍTULO 5. Construcción del modelo de recomendación

Para crear la matriz de interacción playlist/pistas y para crear la matriz de interacción de playlist/artistas, junto a su matriz de pesos, hemos utilizado la función ***build\_interactions***.

Para el caso de la matriz de características de las playlists hemos empleado la función de extracción de características ***LabelBinarizer*** [31] que contiene la biblioteca de *scikit-learn*.







				
John 	1	1	1	1
Tom 	0	0	0	1
Alice 	1	0	1	0

Figura 23. Ejemplo de matriz de interacción [40]

				
John 	5	1	3	5
Tom 	?	?	?	2
Alice 	4	?	3	?

Figura 24. Ejemplo de matriz de pesos de las interacciones [40]

Todo el preprocesamiento que hemos realizado para poder usar el modelo *LightFM* se encuentra en las libretas “*Parte 8 - Preprocesamiento 1*” y “*Parte 9 - Preprocesamiento 2*”.

### 5.3. DEFINICIÓN DEL MODELO

A continuación, vamos a hablar de cómo hemos definido el modelo de *LightFM* que hemos empleado en este trabajo. En la Figura 25 podemos ver qué parámetros recibe la función que se encarga de crear el modelo, junto a los valores que toma por defecto.

En nuestro caso no hemos podido hacer uso de las funciones de ajuste de hiperparámetros, ya que los tiempos de entrenamiento son muy elevados, no podíamos hacer uso de máquinas con mayor capacidad de cómputo (por limitaciones de la suscripción de estudiante) y porque el crédito que tenemos disponible en el proveedor *cloud* es limitado.

```
classlightfm.LightFM(no_components=10, k=5, n=10,  
                     learning_schedule='adagrad', loss='logistic',  
                     learning_rate=0.05, rho=0.95, epsilon=1e-06,  
                     item_alpha=0.0, user_alpha=0.0,  
                     max_sampled=10, random_state=None)
```

Figura 25. Parámetros por defecto para crear el modelo *LightFM*

La decisión que hemos tomado para establecer los parámetros del modelo, ha sido emplear como referencia algunos de los valores que aparecen en el artículo “*Learning to Rank Sketchfab Models with LightFM*” [32].

Los parámetros que hemos definido son los siguientes:

- Función de pérdida (*loss*): WARP (Weighted Approximate-Rank Pairwise). Hemos elegido esta función ya que únicamente tenemos interacciones positivas y pretendemos optimizar la parte superior de la lista de recomendaciones.
- Número máximo de muestras (*max\_sampled*): 30. Hemos elevado el número de muestras para mejorar la precisión del modelo.

## CAPÍTULO 5. Construcción del modelo de recomendación

- Número de componentes (*no\_components*): 200. Se ha elegido este valor de entre los propuestos [32] para que cada item tenga un número de características amplio con las que ser definido, procurando que el tiempo de entrenamiento no sea muy elevado.
- Penalización para las características (*user\_alpha*):  $10^{-6}$ . Se mínima la penalización recomendada.

Para los parámetros restantes, *learning\_schedule* y *learning\_rate*, hemos decidido dejar los valores por defecto (Figura 25).

Llegados a este punto, realizamos el entrenamiento de nuestro modelo con los datos del conjunto de playlists en la máquina virtual, cuyas especificaciones se encuentran en la Tabla 3, que habíamos creado para esta tarea.

El entrenamiento ha requerido un total de 10 horas de ejecución, para completar los 150 ciclos (*epoch*) que hemos establecido, y cuyo modelo resultante tiene un tamaño de 12 GB.

En la libreta *Jupyter* titulada “10 – Definición del modelo *LightFM* y entrenamiento”, podemos encontrar las definiciones de los modelos que hemos empleado junto a los procesos de entrenamiento que hemos llevado a cabo.

# CAPÍTULO 6. EXPERIMENTOS Y RESULTADOS

---

Tras completarse el entrenamiento, vamos a realizar una serie de experimentos en los que comprobamos los resultados que produce nuestro modelo realizando predicciones a las playlists que tenemos en nuestro conjunto de entrenamiento. Una vez realizados dichos experimentos, vamos a emplear las 10.000 playlists incompletas, que habíamos clasificado previamente en 10 categorías, para ver el desempeño de nuestro modelo.

## 6.1. EXPERIMENTOS

Los experimentos que hemos llevado a cabo están divididos en tres partes: experimentos con 3 canciones que conocemos, 3 playlists que hemos seleccionado (cuyo contenido nos es conocido) y 3 playlists aleatorias.

### 6.1.1. EXPERIMENTOS CON PISTAS

En este experimento, hemos seleccionado 3 canciones que conocemos y en las que podemos ver si nuestro modelo nos recomienda canciones que son relacionadas con éstas. Las canciones que hemos seleccionado son las siguientes:

- ‘Toxic’ de *Britney Spears*
- ‘Cool’ de *Alessa*
- ‘Six Feet Under’ de *The Weeknd*

## CAPÍTULO 6. Experimentos y resultados

Tras emplear nuestro modelo para predecir canciones similares, hemos obtenido los siguientes resultados:

<b>*** Toxic (Britney Spears) ***</b>	
Hollaback Girl	Gwen Stefani
Toxic	Britney Spears
Crazy In Love (feat. Jay-Z)	Beyoncé
Don't Stop The Music	Rihanna
Single Ladies (Put a Ring on It)	Beyoncé
Hollaback Girl	Gwen Stefani
Womanizer	Britney Spears
Don't Cha	The Pussycat Dolls
Hips Don't Lie	Shakira
Wannabe	Spice Girls

*Figura 26. Resultados de canciones similares para "Toxic"*

<b>*** Cool - Radio Edit (Alesso) ***</b>	
Love Me Again - Kove Remix	John Newman
When The Bassline Drops (Craig David X Big Narstie	Craig David
Remind Me	High Contrast
Fast Car	Jonas Blue
Desire - Gryffin Remix	Years & Years
With Every Heartbeat - Radio Edit	Robyn
Sorry	Justin Bieber
I'm In Control - The Magician Remix	AlunaGeorge
Middle	DJ Snake
Love You Better	Anton Powers

*Figura 27. Resultados de canciones similares para "Cool"*

<b>*** Six Feet Under (The Weeknd) ***</b>	
Party Monster	The Weeknd
Reminder	The Weeknd
Ordinary Life	The Weeknd
All I Know	The Weeknd
Acquainted	The Weeknd
Sidewalks	The Weeknd
Often	The Weeknd
Crew Love	Drake
Shameless	The Weeknd
Low Life	Future

*Figura 28. Resultados de canciones similares para "Six Feet Under"*

Para la primera pista que hemos empleado como ejemplo, vemos que las canciones que nuestro modelo ha recomendado resultan ser de artistas similares al de la canción consultada. En este caso se da la peculiaridad de que vuelve a aparecer la misma canción que hemos consultado en los resultados. Esto se debe a que una canción puede aparecer en álbumes distintos y en cada uno de ellos *Spotify* la almacena con un identificador distinto.

En el segundo caso, nuestro modelo nos ha devuelto una serie de canciones cuyos estilos musicales son similares al de la canción que hemos elegido.

Para la última canción que hemos consultado, nuestro modelo nos ha devuelto 8 canciones del mismo artista (en las que la mayoría corresponden al mismo álbum) y 2 canciones de artistas similares. Probablemente este caso se deba a que la canción empleada no resulte muy conocida y sólo aparezca en playlists en las que figuren el resto de canciones que corresponden al álbum del artista.

### 6.1.2. EXPERIMENTOS CON PLAYLISTS

Para realizar la primera parte de este experimento, hemos seleccionado 3 playlists que contienen música que nos resulta conocida. A diferencia del experimento anterior, hemos aplicado validación cruzada sobre las playlists, de tal manera que extraemos el 20% de las pistas que contiene para posteriormente ver en qué parte de las  $k$  primeras posiciones aparecen. La métrica que emplearemos en este caso es la *precisión* (*precision@k*, proporción de elementos recomendados en el conjunto top- $k$  que son relevantes). En nuestro caso vamos a consultar la precisión cuando  $k$  es igual a 10, 25, 50 y 100. Las playlist que hemos elegido son las siguientes:

- “Teen” (PID 203663).
- “Kisses For Breakfast” (PID 98312).
- “F.E.E.L. – Lucky Luke” (PID 4569).

En la Figura 29, Figura 30 y Figura 31 podemos observar las 8 primeras pistas que contiene cada playlist, las 8 primeras recomendaciones que ha proporcionado nuestro modelo para completar cada una de ellas y las pistas empleadas para validación.

<b>***PLAYLIST 203663   Teen***</b>	
<b>#Pistas conocidas#</b>	
...Baby One More Time	Britney Spears
Sometimes	Britney Spears
Toxic	Britney Spears
Oops!...I Did It Again	Britney Spears
I'm a Slave 4 U	Britney Spears
Genie in a Bottle	Christina Aguilera
Rock Your Body	Justin Timberlake
Cry Me a River	Justin Timberlake
<b>#Pistas recomendadas#</b>	
No Scrubs	TLC
Waterfalls	TLC
Kiss from a Rose	Seal
Fast Car	Tracy Chapman
I Want It That Way	Backstreet Boys
If I Ain't Got You	Alicia Keys
Always Be My Baby	Mariah Carey
I Want It That Way	Backstreet Boys
Torn	Natalie Imbruglia
Say My Name	Destiny's Child
<b>#Pistas para validación#</b>	
Stop	Spice Girls
What Goes Around.../...Comes Around (Interlude)	Justin Timberlake
Cry Me a River	Justin Timberlake
On The Floor - Radio Edit	Jennifer Lopez
If I Let You Go - Radio Edit	Westlife

*Figura 29. Resultados para la playlist conocida n°1*

<b>***PLAYLIST 98312   Kisses For Breakfast ***</b>	
<b>#Pistas conocidas#</b>	
The Other Side (with MAX & Ty Dolla \$ign)	MAX
We Don't Talk Anymore (feat. Selena Gomez)	Charlie Puth
I Like Me Better	Lauv
High Hopes	Panic! At The Disco
The Other Side	Jason Derulo
Worth It	Fifth Harmony
Work from Home (feat. Ty Dolla \$ign)	Fifth Harmony
I Will Never Let You Down	Rita Ora
<b>#Pistas recomendadas#</b>	
Sunflower - Spider-Man: Into the Spider-Verse	Post Malone
Without Me	Halsey
Eastside (with Halsey & Khalid)	benny blanco
Happier	Marshmello
Sucker	Jonas Brothers
Talk	Khalid
I Don't Care (with Justin Bieber)	Ed Sheeran
bad guy	Billie Eilish
<b>#Pistas para validación#</b>	
Sunflower - Spider-Man: Into the Spider-Verse	Post Malone
7 rings	Ariana Grande
Meant to Be (feat. Florida Georgia Line)	Bebe Rexha

*Figura 30. Resultados para la playlist conocida n°2*



<b>***PLAYLIST 4569   F.E.E.L. - Lucky Luke***</b>	
<b>#Pistas conocidas#</b>	
The Hum	Dimitri Vegas & Like Mike
Virus (How About Now)	Martin Garrix
Body Talk (feat. Julian Perretta)	Dimitri Vegas & Like Mike
Hangover	Dynoro
Drop It	Various Artists
F.E.E.L.	Lucky Luke
Cooler Than Me	Lucky Luke
I Could Be The One (Avicii Vs. Nicky Romero) - Rad	Avicii
<b>#Pistas recomendadas#</b>	
In My Mind	Dynoro
Dancin (feat. Luvli) - Krono Remix	Aaron Smith
Great Spirit	Armin van Buuren
Losing It	FISHER
U Got That	Halogen
Free Tibet - Vini Vici Remix	Hilight Tribe
Tokyo Drift	KVSH
Old Town Road - Remix	Lil Nas X
<b>#Pistas para validación#</b>	
Secrets	Tiësto
Tsunami	DVBBS
Still Cold / Pathway Private	Night Lovell
U Do (feat. Siadou)	TRFN

*Figura 31. Resultados para la playlist conocida nº3*

Si estudiamos con detalle las recomendaciones que ofrece nuestro modelo para completar las playlists seleccionadas, vemos que existe relación entre el contenido de todas ellas y sus títulos. En la Figura 9 podemos ver los resultados de las métricas para las 3 playlists.

	precision@10	precision@20	precision@50	precision@100
<b>Playlist 1</b>	0	0	0	0,400
<b>Playlist 2</b>	0	0	0,250	0,750
<b>Playlist 3</b>	0,167	0,250	0,417	0.625

*Tabla 9. Métricas para las playlists seleccionadas*

Por último, en la Figura 32, Figura 33 y Figura 34 se muestran las recomendaciones ofrecidas por nuestro modelo para completar las 3 playlists que hemos seleccionado de forma aleatoria.

<b>***PLAYLIST 942690   Favourite French Music 🍷***</b>	
<b>#Pistas conocidas#</b>	
Somethin' Stupid	Frank Sinatra
Je veux	Zaz
Les passants	Zaz
Qué vendrá	Zaz
Bonnie And Clyde	Brigitte Bardot
La Javanaise - Mono Version	Serge Gainsbourg
La Madrague	Brigitte Bardot
Comic Strip	Serge Gainsbourg
...	
<b>#Pistas recomendadas#</b>	
Les Champs-Élysées	Joe Dassin
Comment te dire adieu - It Hurts to Say Goodbye	Françoise Hardy
La Vie en rose	Édith Piaf
Douce France	Charles Trenet
La mer	Charles Trenet
Nathalie	Gilbert Bécaud
Quelqu'un m'a dit	Carla Bruni
J'attendrai	Jean Sablon
...	
<b>#Pistas para validación#</b>	
La mer	Charles Trenet
La vie en rose	Édith Piaf
Plus bleu que tes yeux	Édith Piaf
Sylvie	Charles Aznavour
Plus bleu que tes yeux	Charles Aznavour
Mon Raymond	Carla Bruni
Comment te dire adieu - It Hurts to Say Goodbye	Françoise Hardy

Figura 32. Resultados para la playlist aleatoria n°1

<b>***PLAYLIST 826448   u dance therapy***</b>	
<b>#Pistas conocidas#</b>	
Lady Marmalade - From "Moulin Rouge" Soundtrack	Christina Aguilera
One Kiss (with Dua Lipa) - Oliver Heldens Remix	Calvin Harris
Losing It - Radio Edit	FISHER
Le Freak - Oliver Helden's Remix	CHIC
Rhythm Is A Dancer (feat. Kaleena Zanders)	Breathe Carolina
King Of My Castle - Don Diablo Edit	Keanu Silva
The Heat (I Wanna Dance with Somebody)	Ralph Felix
...	
<b>#Pistas recomendadas#</b>	
In My Mind	Dynoro
Giant (with Rag'n'Bone Man)	Calvin Harris
One Kiss (with Dua Lipa)	Calvin Harris
Old Town Road - Remix	Lil Nas X
Promises (with Sam Smith)	Calvin Harris
bad guy	Billie Eilish
Piece Of Your Heart	MEDUZA
Jackie Chan	Tiësto
...	
<b>#Pistas para validación#</b>	
Giant - Robin Schulz Remix	Calvin Harris
Nothing Breaks Like a Heart - Don Diablo Remix	Mark Ronson

Figura 33. Resultados para la playlist aleatoria n°2

<b>***PLAYLIST 229184   DEEP***</b>	
<b>#Pistas conocidas#</b>	
Bon Voyage	DROELOE
Wicked Winds	Mazde
Episode (ANR143) - Dub Mix	Autograf
Sonderling - Radio Edit	Zonderling
Supine	Max Cooper
Nostalgia - Original Mix	Mandragora
Unterwegs	Kölsch
All Eyes On You	Klangkarussell
...	
<b>#Pistas recomendadas#</b>	
Salzburg	Worakls
Purple Noise	Boris Brejcha
Coeur De La Nuit - Worakls Remix	Ferdinand Dreyssig
Reykjavik	Joachim Pastor
Trauma - Worakls Remix	N'to
Angels of Destruction - Neelix Remix	Phaxe
Free Tibet - Vini Vici Remix	Hilight Tribe
The Last Kiss	Querox
...	
<b>#Pistas para validación#</b>	
Butterfly - Ranji Remix	Danny Darko
Reykjavik	Joachim Pastor
Promesse - Short Version	Joachim Pastor
Eche	Joachim Pastor
Atlas - Adriatique Remix	Marc Romboy
Nocturne	Worakls
Sunset - Joris Delacroix Remix	Oliver Schories
Clancy	Oliver Schories
Palm Tree Memories - n'to Remix	Oliver Schories

*Figura 34. Resultados para la playlist aleatoria n°3*

Comprobando los resultados obtenidos tras las predicciones de nuestro modelo, para completar estas 3 playlists aleatorias, podemos observar que el contenido recomendado es similar al de cada una y en el caso de las dos primeras playlists podemos ver que también tienen relación con el título. En la Tabla 10 se muestran los resultados de las métricas empleadas para cada una de las playlists:

	precision@10	precision@20	precision@50	precision@100
<b>Playlist 1</b>	0,167	0,250	0,417	0,625
<b>Playlist 2</b>	0	0	0,500	0,500
<b>Playlist 3</b>	0,689	0,207	0,276	0,345

*Tabla 10. Métricas para las playlists aleatorias*

Tras concluir los experimentos realizados hemos comprobado que los resultados obtenidos, tras las recomendaciones de nuestro modelo para el completado de playlists, son los esperados. En determinados casos las pistas que hemos empleado para validación no se encuentran en el *Top 20*, pero si miramos las recomendaciones ofrecidas podemos ver que las canciones se asemejan lo suficiente con el resto de contenido de las playlists y sus títulos.

Estos experimentos se encuentran en la libreta *Jupyter* titulada “P11 – Experimentos”.

### 6.2. RESULTADOS

Por último, vamos a emplear el conjunto de 10.000 playlists cuyo contenido esta incompleto, y han sido agrupadas en 10 categorías de 1.000 listas, para ver el rendimiento de nuestro modelo ante resultados desconocidos.

En el caso de las métricas de los resultados, volvemos a emplear la precisión cuando  $k$  es igual a 10, 20, 50 y 100.

Antes de ver los resultados obtenidos, vamos a recordar qué categorías tenemos en el conjunto de prueba empleado:

1. Predicción de pistas para una playlist dado sólo su título.
2. Predicción de pistas para una playlist dado su título y la primera pista.
3. Predicción de pistas para una playlist dado su título y las primeras 5 pistas.
4. Predicción de pistas para una playlist dadas las primeras 5 pistas (sin título).
5. Predicción de pistas para una playlist dado su título y las primeras 10 pistas.
6. Predicción de pistas para una playlist dadas las primeras 10 pistas (sin título).
7. Predicción de pistas para una playlist dado su título y las primeras 25 pistas.
8. Predicción de pistas para una playlist dado su título y 25 pistas aleatorias.
9. Predicción de pistas para una playlist dado su título y las primeras 100 pistas.
10. Predicción de pistas para una playlist dado su título y 100 pistas aleatorias.

Una vez concluido el proceso de predicción de pistas para completar las playlists, hemos obtenido los resultados que se muestran en la Tabla 11.

	precision@10	precision@20	precision@50	precision@100
<b>Categoría 1</b>	0,0115	0,0202	0,0394	0,0613
<b>Categoría 2</b>	0,0251	0,0421	0,0802	0,1192
<b>Categoría 3</b>	0,0380	0,0614	0,1168	0,1738
<b>Categoría 4</b>	0,0266	0,0522	0,1062	0,1646
<b>Categoría 5</b>	0,0410	0,0677	0,1319	0,1964
<b>Categoría 6</b>	0,0374	0,0578	0,1283	0,1891
<b>Categoría 7</b>	0,0429	0,0721	0,1366	0,2064
<b>Categoría 8</b>	0,0620	0,1039	0,1863	0,2698
<b>Categoría 9</b>	0,0273	0,0466	0,0959	0,1570
<b>Categoría 10</b>	0,0585	0,0948	0,1724	0,2571

*Tabla 11. Resultados obtenidos en el conjunto de prueba*

Como podemos ver en la tabla anterior, comparando los casos donde se ofrece el mismo número de pistas con título y sin él, categorías 3, 4, 5 y 6, vemos que los resultados de nuestro modelo mejoran sensiblemente si se tiene en cuenta el título.

En las categorías 7, 8, 9 y 10, nuestro modelo es capaz de ofrecer mejores recomendaciones en los casos que se le proporcionan pistas de posiciones aleatorias. Esto se debe a que, en numerosas playlists, las pistas que contienen están agrupadas por artista. Al obtener pistas de esta forma, se proporciona un número de artistas mayor que si recomendásemos las primeras  $n$  posiciones.



# CAPÍTULO 7. CONCLUSIONES Y PROPUESTAS

---

En este capítulo vamos a hablar sobre las conclusiones que hemos obtenido tras haber entrenado los modelos, realizado experimentos sobre determinadas playlists y estudiado los resultados que nuestro modelo ha proporcionado a las playlists que habíamos reservado como prueba.

## 7.1. CONCLUSIONES

A lo largo de este TFG, hemos podido comprobar lo laboriosa que puede resultar la tarea de recopilación y tratamiento de datos, con respecto a las fases de creación y evaluación del modelo de datos. Esto se debe a que hemos partido de cero, sin disponer de un conjunto de datos, además de que nos han ido surgiendo numerosos contratiempos y hemos tenido la necesidad de emplear servicios basados en *cloud* que implican un coste económico. A pesar de ello hemos conseguido definir un proceso de búsqueda, descarga y filtrado de datos con los cuales hemos podido construir un *dataset* de 1 millón de playlists de *Spotify* partiendo de más de 10 millones de listas.

En lo referente a la creación y entrenamiento del modelo, el proceso ha resultado ser menos laborioso que la recopilación de datos, aunque hemos necesitado alquilar una máquina con mayor capacidad de cómputo para desarrollar el trabajo. Pese a no haber podido obtener un conjunto de hiperparámetros óptimos con los que entrenar el modelo, los resultados que hemos obtenido a la hora de predecir recomendaciones de pistas sobre playlists han sido satisfactorios.

A la vista de los resultados obtenidos, podemos concluir que nuestro sistema es capaz de realizar recomendaciones de canciones para completar playlists muy relacionadas a partir del título de ésta y la lista de canciones que la conforman.

Una de las partes más positivas de este trabajo, a parte del resultado que hemos obtenido, ha sido el proceso de aprendizaje de las nuevas herramientas y/o tecnologías que hemos empleado.

### 7.2. PROPUESTAS

Para concluir, vamos a exponer una serie de propuestas que queremos realizar en un futuro para mejorar nuestro sistema de completado de playlists.

#### 7.2.1. OBTENCIÓN DE HIPERPARÁMETROS ÓPTIMOS

Como comentamos a la hora de establecer los hiperparámetros del modelo de *LightFM* que hemos entrenado, al tener un conjunto de playlists tan elevado no pudimos ejecutar algunas de las técnicas que se emplean para tal fin. La idea que se propone es buscar una alternativa que nos permita obtener aquellos valores con los que mejor entrene nuestro modelo, como por ejemplo obteniendo un subconjunto con aquellas playlists más representativas.

#### 7.2.2. CARACTERÍSTICAS PARA ITEMS

Para entrenar el modelo, sólo establecimos las características del usuario (playlists en nuestro caso). Una propuesta que se quiere realizar es obtener características para el conjunto de items (pistas), tales como el año de lanzamiento, las características de audio que proporciona *Spotify* para cada pista [33], etc.



### 7.2.3. MODELOS SECUENCIALES

En nuestro trabajo, no hemos tenido en cuenta la posición en las que se encuentran las pistas de las playlists de nuestro conjunto y tampoco hemos establecido un criterio a la hora de ofrecer las recomendaciones a las playlists.

Recientemente se está experimentando con técnicas que permiten realizar recomendaciones siguiendo la secuencia de interacción del usuario, como por ejemplo “*Translation-Based Recommendation*” [34]. Si tomamos como ejemplo la secuencia de visionado en un servicio de streaming de video como *Netflix*, podemos recomendar a los usuarios qué película pueden visionar a continuación según las secuencias de interacción que hemos obtenido de otros usuarios.

En la Figura 35 podemos ver un ejemplo de recomendación mediante esta técnica para la película *Misión Imposible*.

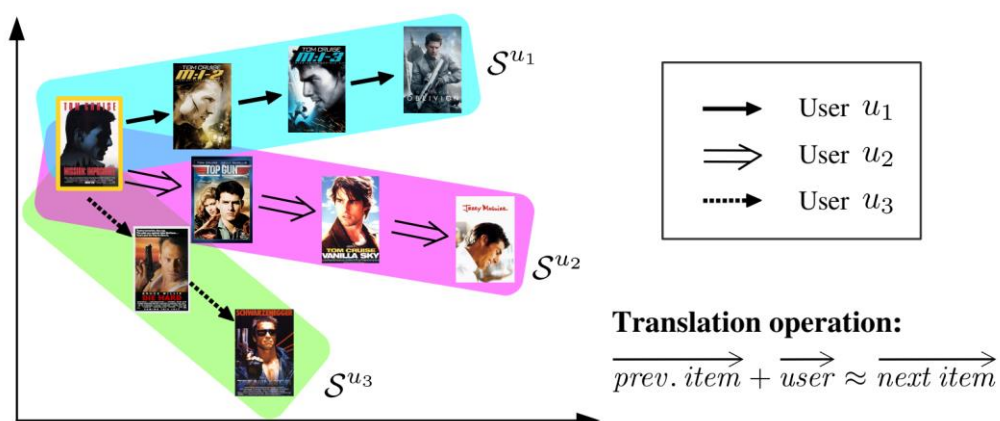


Figura 35. Ejemplo de modelo secuencial [33]

### 7.2.4. DESPLIEGUE DE SERVICIO WEB

Mediante la extracción de la información que ha aprendido nuestro modelo, otra propuesta que se ha considerado es la creación de un servicio web, mediante el cual podamos

## CAPÍTULO 7. Conclusiones y propuestas

ofrecer a usuarios recomendaciones de canciones con las que completar una playlist y que éstos puedan valorar las recomendaciones ofrecidas, empleando un método binario de puntuación (ejemplo: “Me gusta” / “No me gusta”). De esta manera, con el *feedback* que obtenemos de los usuarios, podemos volver a entrenar nuestro modelo de tal manera que tenga información sobre qué tipo de pistas no debe recomendar, estableciendo el valor “-1” para tales casos.

## Bibliografía

- [1] T. McGhie y L. Davison, «Digital music revenues overtake physical sales for the first time,» The Telegraph, 14 Abril 2014. [En línea]. Available: <https://www.telegraph.co.uk/finance/newsbysector/mediatechnologyandtelecoms/11535355/Digital-music-revenues-overtake-physical-sales-for-the-first-time.html>. [Último acceso: 10 Diciembre 2019].
- [2] Spotify AB., «Spotify - Company Info,» 30 Septiembre 2019. [En línea]. Available: <https://newsroom.spotify.com/company-info/>. [Último acceso: 10 Diciembre 2019].
- [3] RecSys Community, «ACM RecSys Challenge 2018,» 20 Noviembre 2018. [En línea]. Available: <http://www.recsyschallenge.com/2018/>. [Último acceso: 11 Diciembre 2019].
- [4] C.-W. Chen, «Introducing The Million Playlist Dataset and RecSys Challenge 2018,» Spotify AB., 30 Mayo 2018. [En línea]. Available: <https://labs.spotify.com/2018/05/30/introducing-the-million-playlist-dataset-and-recsys-challenge-2018/>. [Último acceso: 08 Diciembre 2019].
- [5] S. Huang, «Introduction to Recommender System. Part 1.,» Hacker Noon, 23 Enero 2018. [En línea]. Available: <https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75>. [Último acceso: 14 Diciembre 2019].
- [6] Tryolabs, «Introduction to Recommender Systems in 2019,» 09 Mayo 2018. [En línea]. Available: <https://tryolabs.com/blog/introduction-to-recommender-systems/>. [Último acceso: 14 Diciembre 2019].
- [7] H. Gaspar, «The Cold Start Problem for Recommender Systems,» Yuspify, 14 Julio 2015. [En línea]. Available: <https://www.yuspify.com/blog/cold-start-problem-recommender-systems/>. [Último acceso: 16 Diciembre 2019].
- [8] J. C. L. López, «La moda del Big Data: ¿En qué consiste en realidad?,» elEconomista.es, 27 Febrero 2014. [En línea]. Available: <https://www.eleconomista.es/tecnologia/noticias/5578707/02/14/La-moda-del-Big-Data-En-que-consiste-en-realidad.html>. [Último acceso: 29 Enero 2020].

- [9] Guru99, «Introduction to BIG DATA: What is, Types, Characteristics & Example,» [En línea]. Available: <https://www.guru99.com/what-is-big-data.html>. [Último acceso: 29 Enero 2020].
- [10] J. Ellingwood, «An Introduction to Big Data Concepts and Terminology,» DigitalOcean, 28 Septiembre 2016. [En línea]. Available: <https://www.digitalocean.com/community/tutorials/an-introduction-to-big-data-concepts-and-terminology>. [Último acceso: 29 Enero 2020].
- [11] Naveen, «Introduction to Big Data,» IntelliPaat, 14 Diciembre 2019. [En línea]. Available: <https://intellipaat.com/blog/tutorial/big-data-and-hadoop-tutorial/introduction-to-big-data-2/>. [Último acceso: 29 Enero 2020].
- [12] «MagicPlaylist,» [En línea]. Available: <https://magicplaylist.co/>. [Último acceso: 9 Enero 2020].
- [13] M. Almeida, «Playlist Generator,» Universidad Federal de Minas Gerais, [En línea]. Available: <https://homepages.dcc.ufmg.br/~marcos.almeida/playlistgenerator/index.html>. [Último acceso: 9 Enero 2020].
- [14] V. Øye, «Spotify Playlist Generator,» 9 Agosto 2018. [En línea]. Available: <https://epsil.github.io/spotgen/>. [Último acceso: 9 Enero 2020].
- [15] Osynlig, «Spotlike,» [En línea]. Available: <https://spotlike.com/>. [Último acceso: 09 Enero 2020].
- [16] A. Goicochea, «CRISP-DM, Una metodología para proyectos de Minería de Datos,» 11 Agosto 2009. [En línea]. Available: <https://anibalgoicochea.com/2009/08/11/crisp-dm-una-metodologia-para-proyectos-de-mineria-de-datos/>. [Último acceso: 30 Enero 2020].
- [17] W. Vorhies, «CRISP-DM – a Standard Methodology to Ensure a Good Outcome,» Data Science Central, 26 Julio 2016. [En línea]. Available: <https://www.datasciencecentral.com/profiles/blogs/crisp-dm-a-standard-methodology-to-ensure-a-good-outcome>. [Último acceso: 30 Enero 2020].
- [18] J. V. Román, «CRISP-DM: La metodología para poner orden en los proyectos,» Sngular, 02 Agosto 2016. [En línea]. Available: <https://www.sngular.com/es/data-science-crisp-dm-metodologia/>. [Último acceso: 29 Enero 2020].

- [19] Microsoft, «Azure for Students: crédito de la cuenta gratuita,» [En línea]. Available: <https://azure.microsoft.com/es-es/free/students/>. [Último acceso: 29 Diciembre 2019].
- [20] Spotify AB., «The Million Playlist Dataset README,» [En línea]. Available: <https://recsys-challenge.spotify.com/readme>. [Último acceso: 22 Diciembre 2019].
- [21] Education First, «Lista de las 3000 palabras más usadas en inglés,» [En línea]. Available: <https://www.ef.com.es/recursos-aprender-ingles/vocabulario-ingles/3000-palabras/>. [Último acceso: 20 Diciembre 2019].
- [22] Microsoft, «Text Analytics API | Microsoft Azure,» [En línea]. Available: <https://azure.microsoft.com/es-es/services/cognitive-services/text-analytics/>. [Último acceso: 29 Enero 2020].
- [23] Microsoft, «Detección del idioma con la API Rest de Text Analytics - Azure Cognitive Services,» 30 Julio 2019. [En línea]. Available: <https://docs.microsoft.com/es-es/azure/cognitive-services/text-analytics/how-tos/text-analytics-how-to-language-detection>. [Último acceso: 29 Diciembre 2019].
- [24] Free Web Headers, «Full List of Bad Words and Top Swear Words Banned by Google,» 15 Enero 2019. [En línea]. Available: <https://www.freewebheaders.com/full-list-of-bad-words-banned-by-google/>. [Último acceso: 29 Diciembre 2019].
- [25] H. ZAMANI, M. SCHEDL, P. LAMERE y C.-W. CHEN, «An Analysis of Approaches Taken in the ACM RecSys,» Septiembre 2019. [En línea]. Available: <https://arxiv.org/pdf/1810.01520.pdf>. [Último acceso: 30 Diciembre 2019].
- [26] M. Kula, «LightFM Documentation,» 2016. [En línea]. Available: <https://lyst.github.io/lightfm/docs/index.html>. [Último acceso: 18 Enero 2020].
- [27] M. Kula, «Metadata Embeddings for User and Item Cold-start Recommendations,» 30 Julio 2015. [En línea]. Available: <https://arxiv.org/pdf/1507.08439.pdf>. [Último acceso: 02 Febrero 2020].
- [28] The SciPy Community, «`scipy.sparse.csr_matrix` | ScyPy Docs,» 19 Diciembre 2019. [En línea]. Available: [https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr\\_matrix.html#scipy.sparse.csr\\_matrix](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix). [Último acceso: 02 Febrero 2020].

- [29] The SciPy Community, «[scipy.sparse.coo\\_matrix](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html) | SciPy Docs,» 29 Diciembre 2019. [En línea]. Available: [https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo\\_matrix.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html). [Último acceso: 02 Febrero 2020].
- [30] scikit-learn developers, «[sklearn.feature\\_extraction.text.LabelBinarizer](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html) | scikit-learn API Reference,» [En línea]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html>. [Último acceso: 04 Febrero 2020].
- [31] E. Rosenthal, «Learning to Rank Sketchfab Models with LightFM,» 07 Noviembre 2016. [En línea]. Available: <https://www.ethanrosenthal.com/2016/11/07/implicit-mf-part-2/>. [Último acceso: 04 Febrero 2020].
- [32] Spotify AB, «Get Audio Features for a Track | Spotify Developers,» [En línea]. Available: <https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/>. [Último acceso: 08 Febrero 2020].
- [33] R. He, W.-C. Kang y J. McAuley, «Translation-based Recommendation,» 27 Agosto 2017. [En línea]. Available: <https://arxiv.org/pdf/1707.02410.pdf>. [Último acceso: 09 Febrero 2020].
- [34] N. Routley, «Visualizing 40 Years of Music Industry Sales,» Visual Capitalist, 06 Octubre 2018. [En línea]. Available: <https://www.visualcapitalist.com/music-industry-sales/>. [Último acceso: 11 Diciembre 2019].
- [35] A. Kumar, «Global Online Music Streaming Revenues Cross US\$11 Billion in 1H 2019,» Counterpoint Technology Market Research, 10 Octubre 2019. [En línea]. Available: <https://www.counterpointresearch.com/global-online-music-streaming-revenues-cross-us11-billion-h1-2019/>. [Último acceso: 11 Diciembre 2019].
- [36] R. Channel, «Hybrid recommendation approaches,» 28 Agosto 2017. [En línea]. Available: <https://slideplayer.com/slide/4169010/>. [Último acceso: 15 Diciembre 2019].
- [37] E. Grimaldi, «How to build a content-based movie recommender system with Natural Language Processing,» Towards Data Science, 01 Octubre 2018. [En línea]. Available: <https://towardsdatascience.com/how-to-build-from-scratch-a-content->

- based-movie-recommender-with-natural-language-processing-25ad400eb243.  
[Último acceso: 20 Noviembre 2019].
- [38] J. Desjardins, «What Happens in an Internet Minute in 2019?,» Visual Capitalist, 14 Marzo 2019. [En línea]. Available: <https://www.visualcapitalist.com/what-happens-in-an-internet-minute-in-2019/>. [Último acceso: 29 Enero 2020].
- [39] A. Roy, «Introduction to CRISP DM Framework for Data Science and Machine Learning,» 21 Junio 2018. [En línea]. Available: <https://www.linkedin.com/pulse/chapter-1-introduction-crisp-dm-framework-data-science-anshul-roy/>. [Último acceso: 30 Enero 2020].
- [40] T. Di Noia y V. C. Ostuni, «Example of user-item ratings matrix in a movie recommendation scenario,» Julio 2015. [En línea]. Available: [https://www.researchgate.net/figure/Example-of-user-item-ratings-matrix-in-a-movie-recommendation-scenario\\_fig2\\_300646445](https://www.researchgate.net/figure/Example-of-user-item-ratings-matrix-in-a-movie-recommendation-scenario_fig2_300646445). [Último acceso: 04 Febrero 2020].
- [41] S. Ruder, «An overview of gradient descent optimization algorithms,» 19 Enero 2016. [En línea]. Available: <https://ruder.io/optimizing-gradient-descent/>. [Último acceso: 04 Febrero 2020].





## Contenido del CD

En el CD que acompaña a esta memoria, podemos encontrar los siguientes contenidos:

- Memoria en formato *PDF*
- Código empleado para la realización de este trabajo, almacenado en libretas de *Jupyter*, junto a los archivos requeridos y el fichero de requisitos con las bibliotecas externas de las que hacemos uso.
- Libretas *Jupyter* en formato PDF