Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

COFLATUBE: una base de datos para clasificación de flamenco

Autor: María de las Nieves Fernández Ochoa

Tutor: Jose Miguel Díaz Bañez

Dpto. de Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de la Telecomunicación

COFLATUBE: una base de datos para clasificación de flamenco

Autor:
María de las Nieves Fernández Ochoa

Tutor:
Jose Miguel Díaz Bañez
Catedrático de Universidad

Dpto. de Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: COFLATUBE: una base de datos para clasificación de flamenco

Autor: María de las Nieves Fernández Ochoa
Tutor: Jose Miguel Díaz Bañez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia y a Pedro por apoyarme tanto. No podría no agradecerle a mis compañeros de trabajo Víctor y Antonio por sus ánimos y por sus consejos. A mis amigos de teleco, porque juntos se lleva mejor todo y, como no, a Marta. Por último, agradecerles a Nadine Kroher y Jose Miguel Díaz su paciencia conmigo y los consejos e indicaciones que han hecho posible que este trabajo haya salido adelante.

Sevilla, 2020

Resumen

El flamenco es un género musical cuyo origen se encuentra en Andalucía. Este género está fuertemente caracterizado por la improvisación y la espontaneidad: es por ello que su trasmisión es, sobre todo, de carácter oral. Al igual que con el resto de géneros existentes, el flamenco también ha conseguido, en los últimos años, que su capacidad de acceso mediante archivos digitales (ya sean o no online) aumente, con lo cual se ha hecho evidente la necesidad de describir y analizar su contenido musical automáticamente.

Este trabajo nace con el objetivo de crear una nueva base datos de flamenco, llamada COFLATUBE, que incluya diferentes archivos sonoros provenientes de un ente público como es la Radio y Televisión de Andalucía, la cual lleva realizando, desde hace varios años, un trabajo importante de recopilación y volcado de sus actuaciones de flamenco en una plataforma digital de visualización de contenido como es YouTube. El estudio de estos datos podría indicar cuáles son o han sido los palos o estilos más populares en el mundo flamenco a lo largo de los últimos años.

COFLATUBE ha sido sometida a un procedimiento de minería de datos donde, en primer lugar, se ha realizado un proceso de limpieza, estructuración y normalización de los datos y, posteriormente, se han usado diversos algoritmos que han realizado un análisis estadístico de los datos para facilitar su representación.

Tras el estudio del potencial discriminatorio de COFLATUBE entre los distintos estilos de flamenco, se puede concluir que una de las futuras aplicaciones que podrá tener será la identificación de los diferentes estilos flamencos mediante redes neuronales y aprendizaje profundo.

Abstract

Flamenco is a music genre with its main roots in Andalusia. Flamenco is strongly characterized by improvisation and spontaneity, so it is mostly transmitted by oral means. Just like the rest of existing genres, in the recent years flamenco music has increased its availability within digital files, whether through online resources or not, so the necessity to automatically describe and analyse its content has become evident.

This work has been created with the aim of creating a new flamenco database, called COFLATUBE, which includes sound files from a public entity such as Radio y Televisión de Andalucía, which has done an important work of compilation and publishing of its flamenco performances on a digital online platform such as YouTube. The study of these data could indicate which are or have been the most popular styles in the flamenco world over the last few years.

COFLATUBE has been subjected to a data mining procedure: first, a data cleaning, structuring and a normalization process have been done and, subsequently, various algorithms that perform a stadistical analysis of the data have been used to make their representation easier.

After studying the discriminatory potential of COFLATUBE between the different flamenco styles, it can be concluded that one of the possible applications in the future may be the identification of the flamenco styles through neural networks and deep learning.

Índice

1 Introducción

1.1 Rasgos básicos del cante flamenco

El flamenco es una expresión artística que nació de la mezcla de diversas culturas: la árabe, la judía o la de los gitanos (llegados a Andalucía y otras partes de la Península durante el s.XV) que fomentaron la creación de una inmensa riqueza cultural que tuvo como resultado el nacimiento del Flamenco. Debido a sus características tan particulares e importancia para la identidad cultural del pueblo andaluz, el flamenco forma parte del Patrimonio Cultural Inmaterial de la Humanidad de la Unesco[1] desde 2010.

Dado sus orígenes, las interpretaciones son, en gran medida, fruto de la improvisación, dándose a lo largo de los años una elaborada organización de estilos y estructuras donde se combinan elementos rítmicos y melódicos propios del flamenco junto con un estilo particular propio de cada artista. [3]

La voz es el elemento central del flamenco, generalmente acompañado por la guitarra, baile y palmas, donde se producen cambios repentinos del timbre y del ritmo [8]. Dado el papel tan importante y dominante del canto, éste recibe el nombre, en la jerga flamenca, de *cante*, mientras que sus canciones, *cantes*. El flamenco se organiza en una jerarquía de familias de estilos y sub-estilos denominadas *palos*, los cuales poseen sus propias características mediante secuencias de acordes o patrones rítmicos. [7]. A pesar de la existencia de estas clasificaciones, esta no implica que no sea un terreno vivo y con excepciones continuas, debido a elementos ya mencionados, como la improvisación.

Los elementos o rasgos propios del flamenco son varios [3]:

- **Inestabilidad del pitch**: Las notas se adornan utilizando glissandos, por lo cual el pitch irá variando.
- **Cambios de volumen repentinos**: Se usan como un recurso para generar expresividad.
- **Timbre**: Depende del artista y del cante en cuestión.
- **Inteligibilidad de las voces**: Dada la importancia del cante en el flamenco, es deseable que el público sea capaz de entender las letras con facilidad.
- **Rango melódico corto**: Los cantes suelen girar en torno a una nota, por lo que las escalas utilizadas suelen estar dentro del rango de una octava con respecto a esa nota.

1.2 Music Information Retrieval

Music Information Retrieval (referido a partir de ahora como MIR) es un área de investigación interdisciplinaria emergente que abarca todos los aspectos del acceso a material de música digital [6]. Cada vez es mayor el conocimiento en referencia a la extracción de datos relacionados con la música, además de la consiguiente necesidad de estudiarlos que permita sacar conclusiones importantes en referencia al contenido musical de manera eficiente y efectiva.

[1] Organización de las Naciones Unidas para la Educación, la Ciencia y la Cultura

Durante los últimos años, una variedad de problemas MIR han ido identificándose y, para los cuales, se han propuesto diversas técnicas para resolverlos. Por ejemplo:

- Query by humming: Identificación de melodías cantadas o tarareadas. El sistema buscará en una base de datos de canciones y devolverá al usuario las canciones que contienen el patrón melódico dado.
- Clasificación automática de géneros musicales mediante el análisis de metadatos y extracción de parámetros de la señal de audio.
- Análisis de las estructuras musicales que pueden encontrar encontrar distintos tipos de ocurrencias y correlaciones entre composiciones.
- Obtención del tempo
- Music Generation: Generación automática de música.
- Reconocimiento de instrumentos y separación de una grabación por voces.
- Query by tapping: Se busca el contenido musical que coincida con el patrón generado tras pulsar una tecla o una secuencia de las mismas.

Una de las aplicaciones que más proyección está teniendo es calcular la *huella digital o fingerprint*, que permite identificar cada una de las piezas de una colección o corpus inequívocamente. Es un importante mecanismo de defensa para los derechos de autor. Por ejemplo, es muy interesante esta aplicación en la creación y detección de versiones y covers musicales, que aún se encuentra en fase de desarrollo.

El contenido musical se puede definir como toda información referente a una obra musical que está representada en ella misma. Aquí se distingue entre la señal sonora (serie de valores que representan las oscilaciones de amplitud de la onda sonora a lo largo del tiempo) y aquello que se puede decir al respecto de esa señal una vez escuchada por un oyente [15]. Existen características, consideradas como manuales, las cuales no son parte de la codificación de la pieza en sí, como son el año de la composición, el lugar, el nombre de los artistas, etc. Algunos de estos datos es posible extraerlos computacionalmente con diversas herramientas. A todos estos elementos, que pueden contribuir a definir el contenido sonoro, se les denomina descriptores. Para algunos de ellos es necesario la ayuda de oyentes expertos para la correcta datación de las características de los archivos sonoros. Los descriptores más comunes son los tímbricos, tonales, rítmicos y de género.

Como bien se ha ido comentando, los diferentes archivos sonoros se almacenan en una base de datos junto con la información definida anteriormente como "manual", que ayuda a identificarlos. A este tipo de información se le denomina metadatos [10] y en el caso de querer utilizarlos en el proceso tendrán que pasar por un proceso previo para ser capaces de ser interpretados por un computador.

1.2.1 Estudio computacional del flamenco. Proyecto COFLA

Durante los últimos años, géneros musicales tradicionales ligados al folclore y al desarrollo de la cultura y la vida en los diferentes pueblos del mundo han recibido un reconocimiento creciente en la comunidad MIR y una serie de iniciativas y proyectos de investigación han ido surgiendo con la intención de desarrollar sistemas específicos de análisis para ellos. En este contexto, se desarrolla el interés por el flamenco y el inicio del proyecto COFLA[2].

El proyecto COFLA es un grupo de trabajo multidisciplinar donde expertos de la literatura, matemáticas, ingeniería o psicología musical, colaboran entre sí para aproximarse a una mejor comprensión del flamenco desde todos los ángulos posibles. Algunos de los problemas relevantes en el flamenco y la Flamencología Computacional merecen un enfoque diferente en comparación con otras vertientes musicales. Algunas de las líneas de investigación en las que se está trabajando actualmente son: [3]:

- Transcripción musical de música flamenca: Debido a su trasmisión oral, nunca ha existido la necesidad de trascribir los cantes flamencos; es ahora, con el inicio de su investigación, cuando se ha sentido la necesidad de llevarlo a cabo. Aún no existe una forma consensuada sobre cuál es el mejor método para realizar esta tarea, con lo cual es un terreno que continua abierto a nuevas publicaciones.

[2] Página oficial del proyecto COFLA: *https://www.cofla-project.com/*

- Similitud melódica en la música flamenca: Este terreno resulta realmente complicado de abarcar, debido a la naturaleza del flamenco y sus infinitas peculiaridades. Para el grupo COFLA, ha desembocado en numerosos subproblemas a desarrollar (limpieza de audios y ampliación de los corpus [3]), realización de tests que muestren ausencia de overfitting, segmentación de pitch (contorno melódico), estudio de la voz flamenca, timbre y sistemas de clasificación[8], segmentación de melodías...)
- Detección de patrones melódicos distintivos en la música flamenca: La melodía en el flamenco es de vital importancia, ya que los cantaores han aprendido de memoria cada estilo, al que luego agregarán su toque personal. Hay dos enfoques para abordar este problema: uno de ellos consiste en la realización de diversos análisis computacionales, donde se analiza la presencia de ciertos patrones que aparecen con frecuencia. El otro consiste en definir previamente, gracias a la ayuda de expertos, estos patrones para posteriormente probarlos en la identificación de los mismos en los cantes.
- Clasificación de estilos: Mediante el estudio de diversos parámetros y descriptores se están llevando a cabo algoritmos que, al igual que sucede con otros estilos musicales, faciliten su correcta identificación dentro de los diferentes subgéneros o estilos existentes en el flamenco.

Otros campos de estudio que se abarcan desde el proyecto COFLA son los siguientes:

- Análisis de la estructura métrica.
- Análisis semántico.
- Didáctica: ciencia y música.
- Identificación de cantaores(tanto por audio como por vídeo).
- Música y emoción.

1.3 Justificación y objetivos

Debido a la poca existencia de corpus, en comparación con otros géneros, para el estudio del flamenco, este trabajo pretende crear una base de datos de cantes flamencos (referida a partir de ahora como COFLATUBE) que permita su uso para tareas más complejas dentro del proyecto COFLA. Por ejemplo, COFLATUBE podrá utilizarse para la clasificación de estilos del flamenco mediante algoritmos de aprendizaje profundo o redes neuronales.

A lo largo de este trabajo, se van a utilizar un conjunto de técnicas que se engloban en lo que se conoce como minería de datos. Estas técnicas surgieron con el objetivo de ayudar a comprender grandes masas de datos, las cuales resultan muy complicadas de analizar por su tamaño. Por ello, y con el fin de facilitar el proceso de indexación y clasificación de archivos musicales en el flamenco, se implementarán en COFLATUBE una serie de algoritmos que, paso a paso, irán desgranando el proceso de limpieza, extracción de características y normalización de los datos.

Una vez estos pasos se hayan realizado, se utilizarán herramientas de selección de características que permitan, mediante la asignación de puntuaciones, descubrir cuáles son los descriptores que se consideran más importantes en la clasificación de palos del flamenco, aún teniendo en cuenta la importancia de variables no parametrizables, como la emoción, que juegan un papel fundamental en dicha clasificación.

La representación de los resultados permitirá arrojar ciertas conclusiones sobre la utilidad de estas herramientas para un género tan diverso como el flamenco y podrá abrir camino a nuevas tareas o investigaciones en este ámbito dentro del proyecto COFLA.

2 Recopilación de los datos

En este capítulo se van a detallar los procesos iniciales a la recopilación de los datos, como la preparación del entorno de trabajo y la justificación de la elección del software utilizado. Posteriormente, se seguirán los pasos que componen los procedimientos propios de la minería de datos, como son la extracción de los datos, su limpieza y normalización previa a la aplicación de los algoritmos que permitan extraer información importante para la correcta comprensión de los datos.

2.1 Entorno de desarrollo

1. **Eclipse IDE**: Entorno de desarrollo software construido alrededor de un *workspace* al que pueden incluirse un gran número de plugins, que proporcionan funcionalidades concretas relacionadas con lenguajes específicos o con otras herramientas. Pese a ser un entorno multi-lenguaje, está desarrollado en Java, siendo el desarrollo de este lenguaje su aplicación principal. Eclipse proporciona herramientas para la gestión de espacios de trabajo, escribir, desplegar, ejecutar y depurar aplicaciones. Dado que este trabajo está desarrollado en Python, ha sido necesario la instalación del plugin Pydev, del que se hablará más adelante.

   Algunas de las características principales de Eclipse son:

   - Es una herramienta de código abierto.
   - Salvo el núcleo de la aplicación, todas las funcionalidades de Eclipse están desarrolladas como plugins. Entre sus plugins se encuentran funcionalidades tan útiles como la interacción con repositorios de código compartido como Git o CVS.
   - Es posible el desarrollo de plugins personalizados, así como de aplicaciones para el servidor y servicios web.

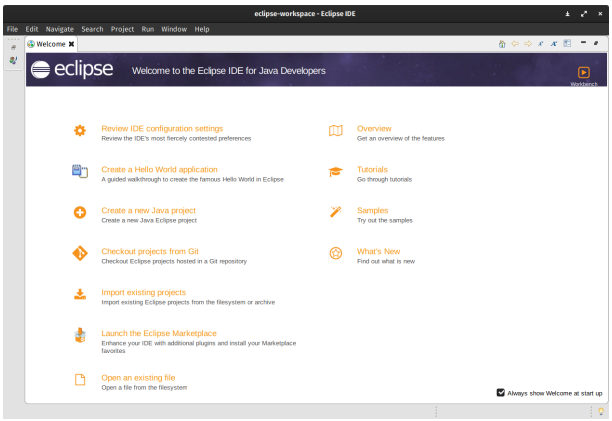

Figura 2.1 Pantalla de bienvenida a Eclipse.

Como se ha comentado anteriormente, para poder trabajar con este entorno de desarrollo en Python es necesaria la instalación de Pydev y posteriormente la configuración del intérprete de Python. Una vez realizado este proceso de instalación, es posible abrir un proyecto de Python.

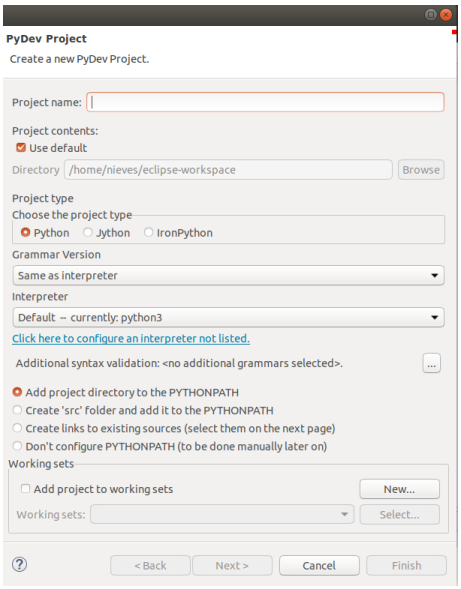

Figura 2.2 Pantalla de creación del proyecto en Python.

2. **Jupyter Notebook**: Es una aplicación web de código abierto que permite al usuario crear documentos que pueden incluir código, widgets interactivos, trazados, textos, ecuaciones o imágenes. En este caso se ha usado para ejecutar código comando a comando de cara a la realización de pruebas concretas sobre el uso de los algoritmos y otras librerías que han sido necesarias.

   Esta aplicación funciona a través de kernels o núcleos, que ejecutan el código de un determinado lenguaje y devuelve una salida al usuario. En el caso que ocupa a este trabajo solo ha sido necesario el uso del kernel de Python.

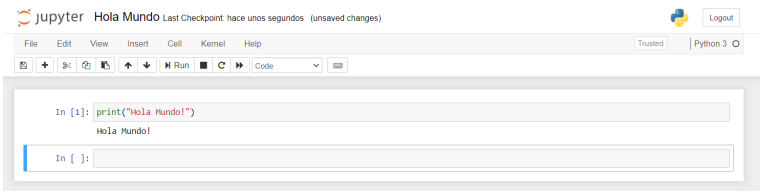

Figura 2.3 Interfaz gráfica de Jupyter Notebook.

3. **Librerías de Python preinstaladas**: Para trabajar con los datos, se han instalado diversas librerías, que se detallan a continuación. Algunas de ellas han sido instaladas por su dependencia con los algoritmos de Machine Learning que se han utilizado.

   - **Numpy**: Agrega a Python mayores funcionalidades para trabajar con vectores y matrices. Por ejemplo, permite trabajar con matrices (arrays) de n dimensiones.
   - **Pandas**: Extensión de Numpy para manipular y analizar de manera sencilla estructuras de datos. Es muy útil para manejar los datos en bruto y proceder a la limpieza de los mismos para que sean aptos para su análisis.
   - **Matplotlib**: Permite la generación de gráficos a partir de datos en listas o arrays. Se abordarán sus funcionalidades más en profundidad en el capítulo de Análisis de los Datos.

- **Scipy**: Paquete científico con diferentes aplicaciones, tales como interpolación, optimización, procesamiento de imágenes, estadística, etc.
- **Joblib**: proporciona herramientas para mejorar el rendimiento y la reproducibilidad con ejecuciones de larga duración.
- **Scikit-image**: Coleción de algoritmos de procesamiento de imágenes. Usa matrices Numpy como objetos de la imagen.

2.2 Definición de COFLATUBE como base de datos

Para la creación de COFLATUBE se ha contado con los contenidos visuales y musicales que RTVA[1] posee en su canal de YouTube llamado Canal Andalucía Flamenco[2]. El canal está formado, en su mayoría, por actuaciones en directo emitidas en televisión por Canal Sur. En el momento del vuelco de los metadatos en un archivo Excel, se cuenta con un total de 1704 vídeos de YouTube. Entre todos sus metadatos,un total de 13, los que van a resultar más interesantes para el análisis son:

- **id**: parte de la URL que identifica al vídeo en YouTube. Servirá para la descarga de los archivos de audio.
- **title**: título del vídeo donde suele ir incluido el autor, el año y el palo flamenco.
- **length**: la duración de la canción en segundos.
- **viewcount**: número de reproducciones del vídeo en YouTube.
- **description**: texto agregado junto al canal con los detalles y datos que RTVA ha estimado necesarios. Suelen incluir más detalles acerca de los intérpretes o del lugar donde tienen lugar las actuaciones, así como el nombre de los programas de televisión.
- **likes**: número de *Me gustas* que ha recibido el vídeo en YouTube.
- **keywords**: palabras que RTVA ha considerado necesario enfatizar de cada uno de los vídeos subidos al canal, de cara a facilitar la búsqueda para los usuarios.

2.2.1 Limpieza de los datos

La limpieza de los datos o 'Data Cleaning'resulta fundamental para trabajar de la forma más rigurosa posible. Frecuentemente, en la primera recopilación de los datos aparecen discrepancias entre los diferentes campos, por ejemplo, en la acentuación de las palabras. Esto puede provocar que se dupliquen los datos o se dividan en categorías diferentes, lo cual haría que se recayese en errores y conclusiones erróneas. Las tareas que comprenden el proceso de limpieza de datos suelen ser la igualación de los formatos, el descarte o borrado de campos y/o columnas que no aportan información para el procesamiento de los datos, o incluso adecuar el formato de las fechas. Si se tienen errores en los datos, se tendrán errores en los resultados.

Para la creación de COFLATUBE, el primer paso llevado a cabo ha sido la eliminación de ciertos vídeos que no incluían contenido musical, como es el caso de vídeos de testimonios, entrevistas, documentales, biografías o programas. Para realizarlo, se ha convertido la estructura de datos en un DataFrame de *pandas*, así será mucho más sencilla la manipulación de los datos. realizado este paso, se guarda en una variable la columna correspondiente a la columna de 'title', que como se ha comentado anteriormente, contiene datos como autor, año o palo flamenco. Es también en esta columna donde se encuentran los campos a eliminar. A través de un bucle `for` y la función `find()` se irán buscando esos campos; si los encuentra, les va a asignar el valor `NaN`. Esto hará posible la eliminación de todas las filas con algún valor nulo mediante la función de la librería pandas `dropna()`. También, observando cada uno de los campos, puede observarse que algunas de las filas no poseen cante, sino bailes o actuaciones donde no aparece ningún cantaor. Para estos casos se ha creado una columna adicional donde se añade esta casuística escribiendo manualmente "NO CANTE". Por otra parte, es importante mencionar la existencia de cantes que no aparecen etiquetados en sus respectivos palos, es decir, que estos palos no se conocen. Para ello, se ha necesitado de la ayuda de expertos en este ámbito para ayudar a identificarlos. Así, este proceso de revisión y etiquetado ha hecho que el proceso de

[1] Radio Televisión de Andalucía
[2] *https://www.youtube.com/channel/UCWyKqd2is0VtWn2qKexkYpw*

limpieza sea semiautomático.

Otra de las tareas que ha ocasionado que el proceso no sea enteramente automático ha sido la revisión de los nombres de los palos. En algunos cantes aparecían escritos de una forma, mientras que en otros, de otra. Es el caso, por ejemplo, de las *seguiriyas*, donde en algunos casos aparece escrito como *seguirillas* o *siguiriyas*. Este hecho puede ocasionar que se dupliquen categorías o clusters, provocando errores como bien se ha ido comentando.

Continuando con el proceso de limpieza, se observa que en los metadatos de la columna 'title'hay bastantes datos que aportan información y que sería útil dividir la columna en varias para tratar cada dato de manera independiente: una columna con el palo llamada 'style', otra para los y las intérpretes que se ha nombrado como 'singer'y, por último, el año de ese cante, denominado como 'year'. Tras esto, se exporta el Dataframe restante a un nuevo archivo Excel. Tras la limpieza, COFLATUBE se ha reducido, en un principio, a un total de 1493 vídeos.

A modo de conclusión, se deja el código comentado:

Código 2.1 CleaningVideo.py.

```
import numpy as np
import pandas as pd
from pandas import ExcelWriter
from pandas import ExcelFile
from pandas._libs.hashtable import nan
from numpy.core.numeric import NaN
from pandas.core.nanops import nanall

#Se carga el archivo excel y se convierte a Dataframe
file = "videos.xlsx";
filePath = "C:/Users/Nieves/eclipse-workspace/tfg2/src/videos.xlsx"
df = pd.read_excel(filePath);
data = pd.DataFrame(df);

#Se guarada en col2 los valores correspondientes a la columna title
col2 = data['title'];

'''Bucle que busca las lineas que tienen las palabras recogidas dentro de
    comando find, les asigna un valor nulo ya que no son cante'''
for i in range(0,len(col2)):
    if col2[i].find("Testimonio")!= -1:
        col2[i]= nan;
    else:
        if col2[i].find("Entrevista")!=-1:
            col2[i]=nan;
        else:
            if col2[i].find("Programa")!= -1:
                col2[i]=nan;
            else:
                if col2[i].find("Perfil") != -1:
                    col2[i]=nan;
                else:
                    if col2[i].find("Biogra") != -1:
                        col2[i]=nan;
                    else:
                        if col2[i].find("Documental") != -1:
                            col2[i]=nan;
```

```
#Se divide por puntos la columna title. Para tener una lista con 3 elementos
data['title']=col2.str.split('.')

''' Se guarda cada elemento de la lista en una variable que formara una tabla
    de elementos '''
t1 = [i[0] if not isinstance(i, float) else None for i in col2]
t2 = [i[1] if not isinstance(i, float) and len(i)>1 else None for i in col2]
t3 = [i[-1] if not isinstance(i, float) and len(i)>2 else None for i in col2]

#Crea las columnas nuevas
data.insert(1, column="style", value=t1)
data.insert(2, column="singer", value=t2)
data.insert(3, column="year", value=t3)

#Se borran las filas con elementos nulos
filasBuenas = data.dropna();

#Elimina la columna title
filasBuenas= filasBuenas.drop('title', 1)

#Se exporta a un archivo excel
export_excel = filasBuenas.to_excel(r'C:/Users/Nieves/eclipse-workspace/tfg2/
    src/videosFinal.xlsx')
```

2.2.2 Representaciones de los datos de COFLATUBE

Si bien es cierto que el contenido facilitado por RTVA continúa creciendo a día de hoy por la subida de nuevos archivos visuales y sonoros, se puede considerar que COFLATUBE es una buena representación de los contenidos de la televisión andaluza a lo largo de los años.

Una vez que la limpieza de los datos se ha realizado, es útil hacer representaciones de los datos que se poseen de cara a tener una visión de conjunto sobre como está formada COFLATUBE. En primer lugar, se ha hecho una representación de la distribución de los cantes por palos así como una tabla resumen con las estadísticas más reseñables.

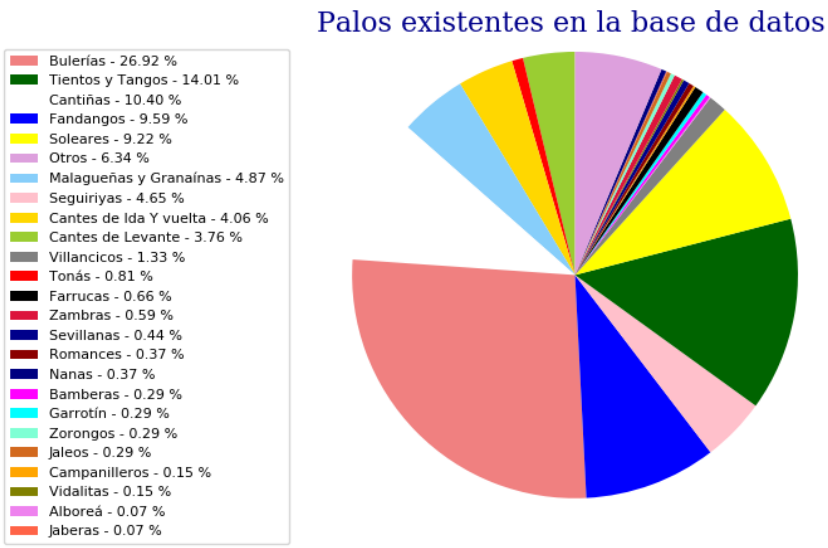

Figura 2.4 Distribución de cantes por palos en la BBDD.

Tabla 2.1 Estadísticas del corpus extraído.

Número total de archivos sonoros	1393
Número de artistas diferentes	624
Duración total en minutos	7759
Intervalo de años de las actuaciones	[1964-2017]
Año con más archivos en el corpus	1991

Como puede observarse los palos más presentes en COFLATUBE son las bulerías, los tientos y tangos, las cantiñas y los fandangos. Es importante recalcar que en esta clasificación no se han hecho distinciones entre palos y subpalos como está recogido en [9]. Esto se puede justificar con la necesidad de tener una visión más generalista de los datos. Para dar una imagen más acorde a esta clasificación se representan, a continuación, las distribuciones de algunos de esos palos junto con sus correspondientes sub-palos. Las características de cada uno de los palos puede consultarse en [5].

- **Cantes de Levante**: Puede observarse como los subpalos más extendidos son las tarantas y los tarantos.

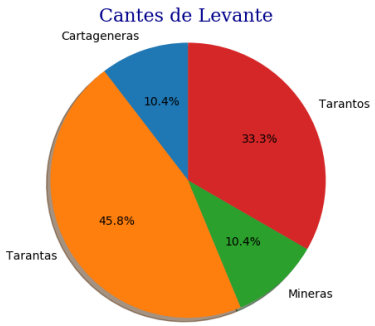

Figura 2.5 Distribución Cantes de Levante.

- **Cantiñas**: Con predominio de las Alegrías, el cante de las fiestas por excelencia, con un compás rápido. Tiene cierto sentido que predomine dentro de las cantiñas en COFLATUBE, ya que en la televisión predominarían las actuaciones con ambiente festivo.

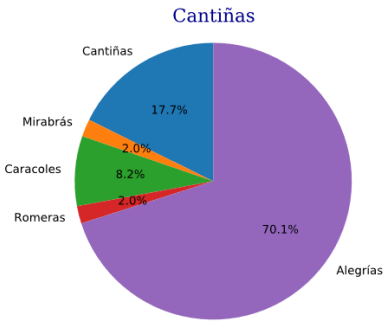

Figura 2.6 Distribución Cantiñas.

- **Fandangos**: Claramente, con predominio de los fandangos. Esto puede deberse a que dentro de los fandangos existen innumerables tipos de fandangos, por ejemplo: fandangos de Alosno, fandangos de Valverde, fandangos de Huelva...

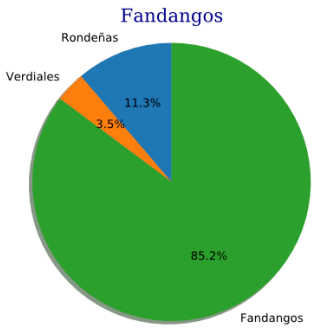

Figura 2.7 Distribución de los Fandangos.

- **Cantes de Ida y Vuelta**: La presencia de los palos está bastante repartida, aunque puede observarse un predominio de guajiras, rumbas y columbianas.

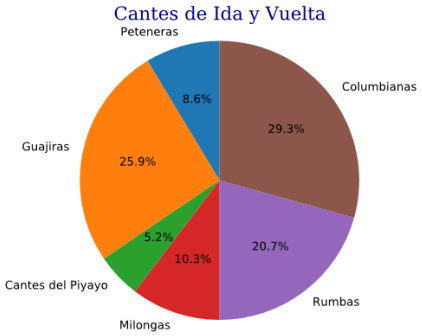

Figura 2.8 Distribución de los cantes de Ida y Vuelta.

- **Seguiriyas**: Prácticamente un 85 % son seguiriyas. Esto se puede explicar de la misma manera que en el caso de los fandangos, donde existen multitud de variaciones. En este caso, se pueden mencionar como ejemplos las seguiriyas de Cádiz o de Jérez. Estos cantes son sombríos y de carácter trágico, con predominio del quejío.

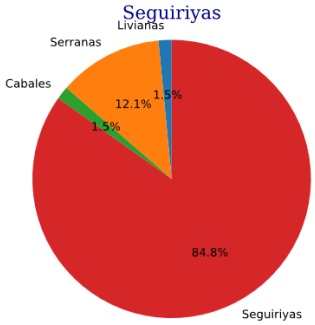

Figura 2.9 Distribución de las Seguiriyas.

- **Soleares**: Los soleares y sus diferentes variaciones predominan frente a las cañas y polos. Este cante está considerado como el centro del arte jondo, de ahí su importancia y su presencia, de casi un 83 %.

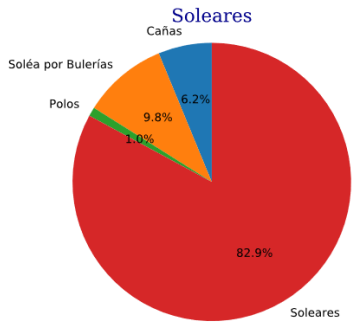

Figura 2.10 Distribución de los Soleares.

- **Tientos y Tangos**: En este caso se ha llevado a cabo la diferenciación de *Tientos y Tangos*, ya que en ocasiones aparecen ambos cantes en un mismo archivo sonoro y en otras ocasiones aparecen separados.

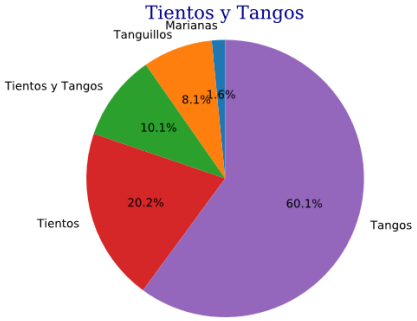

Figura 2.11 Distribución de los Tientos y Tangos.

- **Tonás**: No aparecen ni Deblas ni Carceleras, por lo que solo existen en la BBDD Martinetes y Tonás.

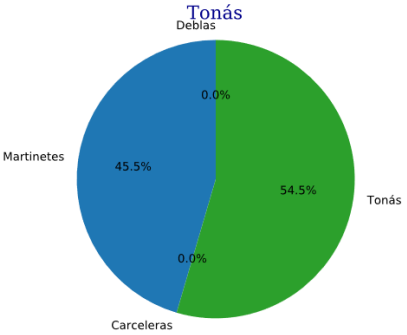

Figura 2.12 Distribución de las Tonás.

Por último, se deja el código de una de las gráficas a modo de ejemplo:

Código 2.2 grafSoleares.py.

```
# -*- coding: utf-8 -*-
import pandas as pd
import matplotlib.pyplot as plt


#Se carga el archivo excel ya limpio y se hace la conversión a Dataframe
file = "videosFinalv1.xlsx";
filePath = "/home/nieves/eclipse-workspace/prueba/prueba/videosFinalv1.xlsx"
df = pd.read_excel(filePath);
data = pd.DataFrame(df);

#Se cuenta el numero de columnas
long=len(data.index);
col2 = data['style'];

caña=0
polo=0
solea_bul=0
solea = 0


#Bucle donde se visualiza el total de cantes por palo para ver la distribución
    de los suppalos
for i in range(0,len(col2)):
    col2[i] = col2[i].lower();
    if col2[i].find("caña")!=-1:
        #print(col2[i])
        caña = caña +1;
    if col2[i].find("polo")!=-1:
        polo=polo+1;
    if col2[i].find("soleá por buler")!=-1 :
        solea_bul = solea_bul+1;
    if col2[i].find("sole")!=-1 :
        solea = solea+1;


#Representación de los datos
labels = 'Cañas' , 'Soléa por Bulerías', 'Polos', 'Soleares'
sizes = [caña, solea_bul,polo, solea ]

font = {'family': 'serif',
        'color': 'darkblue',
        'weight': 'normal',
        'size': 16,
        }
fig1, ax1 = plt.subplots()
plt.title('Soleares', fontdict=font)
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=False, startangle=90)
ax1.axis('equal') # Relación de aspecto para que se dibuje como un circulo
plt.show()
```

2.3 Obtención de los archivos de audio mediante Youtube-dl

Una vez ya se ha finalizado con el proceso de limpieza y representación de los datos, es momento de obtener los datos sonoros de los metadatos que conforman COFLATUBE. Esta acción se llevará a cabo mediante el uso de la librería de Python **Youtube-dl**, que permite descargar el contenido de varios sitios web, entre los que se encuentra YouTube. Es de código abierto y para su uso es necesario tener instalado el intérprete de Python, ya sea para las versiones 2.6, 2.7 o más de 3.2. Además, se ha necesitado de la instalación de **FFmpeg**, que es un framework ampliamente utilizado para la codificación y conversión de audio y vídeo. Más concretamente, para este trabajo se ha utilizado para la conversión del archivo de vídeo en archivos de audio únicamente y en formato mp3.

2.3.1 Cómo funciona

Tal y como aparece reflejado en [13], hay multitud de opciones que pueden utilizarse con Youtube-dl: Se pueden configurar opciones de red, como por ejemplo la elección del tipo de IP o la posibilidad del uso de un proxy, también se pueden introducir restricciones de geolocalización, según el país o la IP, o de autenticación, mediante usuario y contraseña. En cuanto al contenido se refiere, las opciones se encuentran clasificadas en varios grupos. En concreto, las que más interesan son las que se refieren a las opciones de selección y formato de vídeo, las opciones de descarga, opciones de preprocesado o las opciones para gestionar el sistema de archivos. Este herramienta tiene una estructura simple y dividida en cuatro componentes:

- **Youtube-dl, el núcleo de la aplicación**: Responsable del proceso general, que incluye el procesamiento de línea de comandos analizando los argumentos.
- **Los extractores**: recopilan la información del vídeo, como por ejemplo la URL.
- **El componente de descarga**: Se encarga de transferir el vídeo al sistema de archivos local. Según el formato que tenga, utilizará un tipo de descarga u otra.
- **Post-procesadores**: responsable de las operaciones que se realizan después de la descarga, como extraer el audio o incrustar subtítulos.

Como puede observarse, cada uno de los componentes ofrece funcionalidades diferentes asociadas a las opciones anteriormente citadas. Este tipo de estructura, sumado a que el código es accesible para todo el mundo, hace posible que sea sencilla la inclusión de nuevas opciones o actualizaciones a la herramienta. Para más detalles sobre la estructura, es útil la consulta en [14].

En cuanto a FFmpeg [4], aunque está desarrollado en el sistema operativo GNU/Linux, es posible su uso también en Windows. FFmpeg utiliza una librería de codecs llamada libavcodec, de la que pueden hacer uso 3 aplicaciones: ffplay.exe (reproductor multimedia), ffprobe.exe (herramienta de análisis de archivos multimedia) y ffmpeg.exe, que es la herramienta de codificación y conversión de audio y vídeo de la que es dependiente Youtube-dl y de la que se ha hecho uso. El proceso que sigue un archivo de entrada hasta su conversión en un nuevo archivo de salida consiste en la llamada a una librería que demultiplexa la señal de audio para conseguir datos codificados. Para evitar errores, ffmpeg posee un mecanismo para mantener los datos sincronizados haciendo uso de marcas de tiempo. Los paquetes de datos creados se decodifican en frames sin comprimir (raw video) que se van a procesar mediante una cadena de filtros de otra de las librerías incluidas en este framework, llamada libavfilter. Tras el filtrado de los frames, estos se codifican formando paquetes, se multiplexan y se genera el archivo de salida con el formato y características deseadas.

En la siguiente sección, se detalla el proceso junto con las opciones de formato utilizadas, así como las opciones para guardar los archivos de audio.

2.3.2 Aplicación del algoritmo

Para automatizar la descarga de todos los datos que conforman COFLATUBE y evitar el uso de la línea de comandos de forma manual, hecho que ralentizaría considerablemente el proceso, se ha desarrollado un script de Python. Tras la conversión a DataFrame del archivo Excel, ya es posible trabajar con los datos, algo que ya se ha explicado con anterioridad. Para el control de errores o warnings, se ha definido una clase llamada `MyLogger(object)` con varias funciones, que en caso de que ocurran no provocarán que pare la ejecución del código.

Seguidamente, se define una función `my_hook()`, que servirá para mostrar por pantalla el estado del proceso de descarga e inicio del proceso de conversión mediante ffmpeg. Esto se mostrará mediante una sentencia `if` que detectará si el estado está `'finished'`. A continuación, se crea un bucle `for`, que será la parte más importante del código, donde se va recorriendo la columna de las URL. Dentro de este bucle, existe una variable llamada `ydl_opts`, donde se detallan las opciones de formatos, calidad del audio, así como el directorio donde se va a almacenar. Concretando en cada una de las opciones:

- **format**: Se especifica el formato del audio. Se elige la opción `bestaudio`, que selecciona la mejor calidad de audio que sea posible obtener.
- **outtmpl**: Directorio de almacenamiento con el que se desea que se guarde el archivo. Interesa que se almacene con el nombre de la URL y no con el título de la canción, ya que es posible que posteriormente haya que volver a modificar el Excel para mejorar el proceso de data cleaning e identificación de cada uno de los cantes. La URL no va a tener que modificarse en ningún momento, ya que es una característica única de cada uno de los elementos existentes en la BBDD.
- **postprocessors**: Se introducen detalles para el postprocesado del archivo. En el caso que ocupa este apartado se selecciona ffmpeg como `key`, el formato mp3 como codec en `preferredcodec` y, por último, la elección de la calidad en `preferredquality`.
- **logger**: Se selecciona la clase `MyLogger()` creada con anterioridad.
- **progress_hooks**: Con la llamada a la función `my_hook`.

Por último, se llama a Youtube_dl con estas opciones para que las utilice en el proceso de descarga. Es importante mencionar que, pese a que se ha automatizado el proceso, este sigue siendo costoso en lo relativo al tiempo y es totalmente dependiente de la conexión a Internet, además de ser susceptible a varios tipos de errores graves que hacen que se paralice la ejecución y haya que volver a iniciarla. Para ello, se imprime por pantalla, también, la variable i de control del bucle para saber cuál ha sido el último último archivo descargado.

Los errores que se han tenido durante la ejecución están prácticamente todos relacionados con la descarga masiva de archivos. El `HTTP error 402` tiene lugar cuando YouTube pide que se supere un CAPTCHA.Este es un error que los desarrolladores aún no han podido solucionar, aunque sì han dado un procedimiento para resolverlo temporalmente. Otro de los errores comunes que han tenido lugar son `HTTP Error 429: Too Many Resquests` y `402: Payment Required`, que indican que se ha procedido a bloquear la IP por exceso de uso. Volviendo a ejecutar el código se resolvería el problema. Por último, otro de los errores sufridos en este proceso ha sido `ERROR: unable to download video`, que guarda relación con que el usuario haya borrado vídeos del canal, los cuales habrían de descartarse de COFLATUBE. El código completo se muestra a continuación:

Código 2.3 descargaBBDD.py.

```
import os
import numpy as np
import pandas as pd
import youtube_dl
from pandas import ExcelWriter
from pandas import ExcelFile

#Se carga el archivo Excel y se convierte a Dataframe
file = "videosFinalv1.xlsx";
filePath = "C:/Users/Nieves/eclipse-workspace/tfg2/src/videosFinal.xlsx"
df = pd.read_excel(filePath);
data = pd.DataFrame(df);

#se guarda en col1 el contenido de la primera columna que son las URL
col1 = data['id'];

#Clase con funciones en caso de warning o errores
```

```
class MyLogger(object):
    def debug(self, msg):
        pass

    def warning(self, msg):
        pass

    def error(self, msg):
        print(msg);

#Función que muestra el proceso de descarga por pantalla
def my_hook(d):
    if d['status'] == 'finished':
        print('Done downloading, now converting ...');

#Bucle donde se lanza el comando de descarga con opciones de descarga y
    conversión del audio
for i in range(0,len(col1)):
    co= "https://www.youtube.com/watch?v=" + col1[i];

    ydl_opts = {'format': 'bestaudio/best',
                'outtmpl':'C:/Users/Nieves/eclipse-workspace/tfg2/src/BBDD/%(
                    title)s.%(id)s.%(ext)s.',
    'postprocessors': [{
        'key': 'FFmpegExtractAudio',
        'preferredcodec': 'mp3',
        'preferredquality': '192',
    }],
    'logger': MyLogger(),
    'progress_hooks': [my_hook],}
    with youtube_dl.YoutubeDL(ydl_opts) as ydl:
        ydl.download([co]);
```

2.4 Extracción de características mediante Essentia

Para la obtención de los descriptores del corpus creado se ha utilizado una nueva librería llamada **Essentia** [2]. Su creación y uso estuvieron motivados por el contexto de crear un entorno que facilitase a los investigadores los trabajos relacionados con tecnología musical, así como optimizar la ejecución para grandes bases de datos de archivos de audio. Essentia es de código abierto y contiene multitud de algoritmos y distintas funcionalidades para analizar y extraer características de todo tipo, como, por ejemplo, descriptores tonales o rítmicos. Su desarrollo está enfocado en conseguir que sea una herramienta robusta, con un buen rendimiento y que utilice algoritmos que estén optimizados; ya sea en relación al uso de la memoria, la velocidad de cálculo e incluso su facilidad de uso. Por otra parte, esta librería de C++ posee enlaces con Python y JavaScript, así como varias herramientas en línea de comandos y diversas extensiones.

Para este trabajo, el objetivo en este punto es extraer, de cada uno de los elementos del corpus generado en COFLATUBE, características que permitan analizar y poder entrenar, posteriormente, a los algoritmos de Machine Learning para la clasificación de los palos. La herramienta de Essentia que se ha utilizado es un programa ejecutable por línea de comandos llamado `essentia_extreaming_extractor_music`, que calcula un gran conjunto de descriptores espectrales, de dominio del tiempo, de ritmo, tonales y alto nivel y que está especialmente indicado para grandes cantidades de archivos, ya que es la forma más fácil de obtener descriptores sin programar.

2.4.1 Cómo funciona

El diseño general de la arquitectura que conforma Essentia es bastante simple. Cada bloque de procesamiento o algoritmo tiene 3 tipos de atributos: las entradas, las salidas y los parámetros que se le añaden. El número de estos atributos puede variar o incluso no existir alguno de ellos. Estos algoritmos, además, son una subclase de `Configurable`, que es quien se encarga de añadirle los parámetros correspondientes a sus subclases mediante el métodos, `declareParameters()`. Los parámetros pueden representar cualquier tipo de dato (`string`, `integer`, `float`, `boolean`, etc). En cuanto a las entradas y salidas, estas son gestionadas por la clase `Algorithm` mediante los métodos `declareInput()` y `declareOutput()`. Es importante mencionar que estos métodos no almacenan datos, solo apuntan a ellos. Para el almacenamiento, Essentia utiliza unas estructuras a las que denomina `Pool`, que funcionan como una memoria caché.

En cuanto a `essentia_extreaming_extractor_music`, es posible personalizar los parámetros de análisis de audio, los clasificadores de alto nivel o el formato de salida. En este caso, este será un archivo JSON[3] por cada archivo de audio descargado con Youtube_dl. Para extraer las características de cada uno de los archivos de COFLATUBE, se ha optado por usar la ejecución predeterminada de esta herramienta, extrayendo todos los descriptores posibles ya que posteriormente se analizarán las más relevantes. Es importante mencionar que todos los descriptores, por defecto, son analizados con una frecuencia de muestreo de 44kHz.

Hay tres grupos donde pueden dividirse los descriptores de audio:

- **De bajo nivel**: El objetivo de estos descriptores es proporcionar información básica de la señal que haga definir otras herramientas a nivel superior. Algunas de estas características se expresan en escala logarítmica para reflejan la respuesta del oído humano. Por ejemplo, se calculan los silencios en el archivo de audio, los rangos dinámicos y su complejidad, energías espectrales para diferentes bandas de frecuencias, etc. También se calculan diversas estadísticas de medias y varianzas.
- **Rítmicos**: Estos descriptores se centran en el calculo de los BPM (Beats per Minute) desde distintas perspectivas.
- **Tonales**: Se estima, por ejemplo, la escala, los acordes o el pitch.

Para más detalles acerca de los descriptores existentes es muy esclarecedora la consulta a su página web[4], donde se describen y especifican cada una de los elementos de salida del algoritmo que se ha utilizado.

2.4.2 Aplicación del algoritmo y extracción de las características en archivos JSON

El script `ExtractCharacteristics.py` se encarga de llamar al algoritmo de Essentia, proporcionándole una entrada, que será el archivo mp3 descargado y referenciado en el código mediante la variable `cancion`. Una salida en formato json es generada con el nombre indicado en la variable `datos`. Este proceso se va a llevar a cabo para cada uno de los elementos del corpus, gracias a la implementación de un bucle `for`. Dentro de este bucle y para llamar a la línea de comandos, se utiliza la librería de python de `subprocess` [12], que es un módulo con varias interfaces de programación que permite iniciar procesos y ejecutar comandos externos. La función utilizada ha sido `call()` , que aunque viene desde versiones anteriores de Python, sigue siendo compatible con Python 3.6. Introduciendo el comando, la entrada y la salida, además de indicar que el comando se ejecute a través del shell, el script irá generando uno a uno los 1393 archivos JSON correspondientes.

El código detallado se deja a continuación:

[3] JavaScript Object Notation
[4] *https://essentia.upf.edu/streaming_extractor_music.html*

Código 2.4 ExtractCharacteristics.py.

```
import pandas as pd
import subprocess

#se carga el archivo Excel como Dataframe
file = "videosFinalv1.xlsx";
#filePath = "C:/Users/Nieves/eclipse-workspace/tfg2/src/videosFinalv1.xlsx"
filePath = "/home/nieves/eclipse-workspace/prueba/prueba/videosFinalv1.xlsx"
df = pd.read_excel(filePath);
data = pd.DataFrame(df);

#Se cuenta el numero de columnas, además se guarda la columna de las URL en una
     variable
long=len(data.index);
col2 = data['id'];

#Se crea un bucle para tratar cada uno de los archivos
for i in range(0,len(col2)):
    print(i)
    #Se guarda en una variable el nombre completo de los archivos guardados en
        el directorio BBDDv1
    cancion = "BBDDv1/_."+ col2[i]+".mp3 "
    print(cancion)
    #Se crea una variable para indicar el nombre del archivo donde almacenarán
        los datos por canción
    datos = "json/"+col2[i]+ ".json"
    print(datos)
    #Se ejecuta el subprocess que ejecuta streaming_extractor_music
    subprocess.call(["./streaming_extractor_music " +cancion+" "+ datos], shell
         =True)
```

Tras la creación de todos los archivos, se han volcado en un único fichero de formato `csv`[5] para que su tratamiento posterior sea más sencillo y rápido. Para ello, se ha elaborado un script en Python. Como aspectos novedosos con respecto a anteriores códigos desarrollados, destaca la utilización de `json_normalize`, que hace posible tratar los datos JSON que están estructurados en un texto plano y transformarlos a un DataFrame. Esta función se encuentra dentro de un módulo de la librería `pandas`, que ya se ha utilizado en todos los desarrollos anteriores. Para más información acerca de esta funcionalidad, puede consultarse la documentación en su página oficial [6].

También se ha utilizado la librería `json` para cargar los archivos mediante la función `load()` y así poder usarlos a lo largo del código. En el script se han desarrollado dos bucles diferentes: uno de ellos es solo para el primer elemento que se ha asignado a un Dataframe y se ha guardado en una fila. Para poder ir añadiéndole nuevas entradas al Dataframe sin duplicarlas, se ha creado un segundo bucle con un segundo Dataframe `df2`, que se irá borrando con cada iteración, no sin antes añadirse mediante el método `append()` al primer Dataframe. Una vez recorrido todo el bucle, se llama a otro método de la librería `Pandas` utilizado para convertir los Dataframe en archivos `csv`.

Abriendo el archivo `csv` generado tras la ejecución del código, puede observarse como se poseen un total de 1493 filas con 553 características o features en cada una de ellas, lo que se corresponde con lo esperado.

Se recoge todo lo explicado en el código que aparece a continuación:

[5] Tipo de documento que representa datos en forma de tabla, las columnas se separan por comas y las filas por saltos de línea.
[6] *https://pandas.pydata.org/pandas-docs/version/0.21.1/generated/pandas.io.json.json_normalize.html*

Código 2.5 ExportCSV.py.

```
import pandas as pd
from pandas.io.json import json_normalize
import json

#Se pasan todos los datos de los archivos JSON a un csv para facilitar el uso
    de los mismos

#se carga el archivo Excel como Dataframe
file = "videosFinalv1.xlsx";
#filePath = "C:/Users/Nieves/eclipse-workspace/tfg2/src/videosFinalv1.xlsx"
filePath = "/home/nieves/eclipse-workspace/prueba/prueba/videosFinalv1.xlsx"
df1 = pd.read_excel(filePath);
enlaces = pd.DataFrame(df1);
long=len(enlaces.index);
col2 = enlaces['id'];

#Bucle donde se abre el primer archivo json en modo escritura y se guarda en un
     dataframe
for i in range(0,1):
    with open('json/'+col2[0]+'.json', 'r') as write_file:
        carga = json.load(write_file)
        #Convierte el json a Dataframe de pandas
        df = pd.DataFrame.from_dict(json_normalize(carga), orient='columns')

#Bucle donde se recorre el resto de archivos y se van añadiendo a otro
    Dataframe para posteriormente unirlos
for j in range(1,len(col2)):
    with open('json/'+col2[j]+'.json', 'r') as write_file1:
        carga1 = json.load(write_file1)
        df2 = pd.DataFrame.from_dict(json_normalize(carga1), orient='columns')
        df = df.append(df2, ignore_index=True)

print("Creando archivo csv...")
#Se crea el archivo csv
export_csv = df.to_csv (r'/home/nieves/eclipse-workspace/prueba/prueba/
    export_dataframe.csv', index = None, header=True)

print("Archivo csv creado!")

print(df)
```

3 Tratamiento de los datos

Este capítulo está dedicado a todas las tareas y procesos llevados a cabo en los datos obtenidos en el capítulo anterior. En primer lugar, será necesaria una normalización de los datos para que se encuentren, todos ellos en la misma escala y, de esa forma, sea posible compararlos. Este proceso puede ser uno de los más costosos, en cuestión de tiempo sobre todo, de todo los englobado en este trabajo, ya que ha sido necesario adaptarse a diferentes casuísticas según los tipos de datos existentes. Posteriormente, para obtener una selección de las características más importantes, se utiliza uno de los algoritmos de minería de datos y, tras este otro, (ambos procedentes de la librería `scikit-learn`), que servirá para obtener un espacio en dos dimensiones que permita su representación y se puedan visualizar los clusters de cada uno de los palos presentes en COFLATUBE.

3.1 Normalización

Uno de los procesos fundamentales para proceder a la comparación de datos (si éstos se encontrasen en diferentes escalas de magnitud) es la normalización: por ejemplo, resulta complicado poder comparar valores logarítmicos con aquellos de escala decimal. Además, no solo importa la escala a la que se encuentren, sino el tipo de datos. Observándolos, puede verse como se tienen valores tipo float, arrays de float e incluso cadenas de caracteres. Para ello, será necesario el uso de diversas técnicas para tratar de "eliminar" la unidad de medida y que el tipo de datos sea lo más uniforme posible.

3.1.1 Cómo normalizar

Para la realización del proceso principal, es decir, para la mayoría de los valores excepto para los strings y las variables booleanas se ha tomado como referencia a [1], en concreto, el capítulo 3, donde se realiza la normalización mediante el cálculo de diversas estadísticas que originan que se generen nuevos valores de media igual a 0 y desviación estándar igual a 1. El primer paso a implementar es el cálculo de la desviación estándar que, cabe recordar, expresa la dispersión de los valores en torno a su media. De esta forma,se ha definido un conjunto de N valores, cada uno de ellos denotado como Y_N. Con estos elementos, la desviación estándar se puede expresar como:

$$\hat{S} = \sqrt{\frac{\Sigma(Y_N - M)^2}{N-1}} \tag{3.1}$$

Una vez se han obtenido las desviaciones típicas con 3.1, ya es posible normalizar mediante la expresión:

$$Z_n = \frac{Y_N - M}{\hat{S}} \tag{3.2}$$

Este proceso no puede llevarse a cabo para los descriptores que son strings; por ello, será necesario un tratamiento especial. Primeramente, y tras una observación de los datos, se puede ver como existen strings que no aportan información relevante para el contenido musical: son el caso de los descriptores que recogen información sobre la versión de Essentia, el codec del archivo, etc, es decir, son metadatos que no ayudarán a la identificación de los palos. También se pueden ignorar los features relacionados con el "key", ya que normalmente se adaptan al rango de voz del cantante. Por todo ello, quedarán únicamente dos columnas

de strings que están relacionadas con el modo, minor y major, que se hará que sean iguales a 1 o a 0 para convertirlos en caracteres numéricos.

3.1.2 Algoritmo desarrollado para normalizar

El script realizado a continuación viene a reflejar cada uno de los pasos descritos en la sección anterior para la normalización de los datos. Sin duda, es uno de los códigos más extensos que se han implementado, debido principalmente a la variedad de los tipos de datos. Un primer paso en el script consistirá en la eliminación de los features, que no aportan información o son redundantes. Esta operación se ha realizado mediante el método `drop()`, perteneciente a la librería `pandas` junto con la opción `axis = 1` , que indica que se haga por columnas, y por ello se elimine al completo. Tras esto se han creado dos Dataframe, donde se almacenará a lo largo del código los nuevos valores obtenidos para las columnas que son arrays de arrays (`new_num`) y también los de la normalización de todos los datos en `norm_df`. Dicho esto, se crea un bucle donde se van recorriendo, en primer lugar, los arrays de arrays que en el Dataframe son de tipo `object`. A estos valores se les eliminará el primer y último elemento que representan los corchetes y, posteriormente, se separarán en elementos diferentes mediante el método `split()`, con la coma como elemento separador. Una vez realizados estos dos pasos, se van a convertir en una lista de float, algo que facilitará el cálculo de operaciones matemáticas y estadísticas. Como se quiere obtener un único valor por celda y este tipo de columnas tienen varios, se calcula la media de todos los valores para obtener un único valor, que será el que represente el valor de la celda "original". Una vez todos los valores de las columnas están representados con un único valor, ya se puede calcular la media y la desviación típica de la columna al completo: esto se realiza gracias a los métodos `mean()` y `std()`.

Tras las operaciones realizadas, ya es posible aplicar en otro bucle anidado la ecuación 3.2 para normalizar y almacenar los resultados en el Dataframe `norm_df` mencionado con anterioridad. No puede olvidarse que las columnas que representan el modo y que han sido convertidas a 0 y 1 no se han normalizado y, por ello, habrá que añadirlas también a este Dataframe.

Por último, se ha observado que diferentes columnas quedaban en blanco o nulas. Esto ha ocurrido porque los valores de las columnas tenían exactamente el mismo valor en todas sus celdas, por lo que el resultado sería igual a 0, y por tanto, la normalización resultaría en una división por 0. Debido a este hecho, se ha tenido que añadir al script código para detectar estas columnas y eliminarlas, ya que tampoco van a aportar a la identificación de los palos.

Código 3.1 AddDataframeAndNormalize.py.

```
import numpy as np
import pandas as pd

#Se carga el excel y el csv generado con los json

file = "videosFinalv1.xlsx";
#filePath = "C:/Users/Nieves/eclipse-workspace/tfg2/src/videosFinalv1.xlsx"
filePath = "/home/nieves/eclipse-workspace/prueba/prueba/videosFinalv1.xlsx"
df1 = pd.read_excel(filePath);
enlaces = pd.DataFrame(df1);
long=len(enlaces.index);
col2 = enlaces['style'];

df2 = pd.read_csv('export_dataframe.csv')
data  = pd.DataFrame(df2)

norm_df = pd.DataFrame() #dataframe donde se almacenan los valores normalizados
     de todas las columnas
new_num = pd.DataFrame() # dataframe para guardar los valores unicos en las
    columnas que son arrays
```

```
#Se borran las columnas que no son de utilidad y se transforman las dos
    columnas de strings relacionadas con el modo
data['tonal.chords_scale'] = data['tonal.chords_scale'].apply(lambda x:1 if x
    =="major" else 0)
data['tonal.key_scale'] = data['tonal.key_scale'].apply(lambda y:1 if y=="major
    " else 0)

data =data.drop(['tonal.key_key','tonal.chords_key'] , axis=1)

vector_eliminar = ['metadata.audio_properties.codec', 'metadata.
    audio_properties.downmix','metadata.audio_properties.md5_encoded', '
    metadata.tags.file_name',
         'metadata.tags.encoding','metadata.version.essentia','metadata.version
             .essentia_git_sha', 'metadata.version.extractor',
         'lowlevel.silence_rate_60dB.median','lowlevel.silence_rate_30dB.min',
             'lowlevel.silence_rate_20dB.min' ]
data = data.drop(vector_eliminar, axis=1)

#Hay 4 columnas que se eliminar que poseen información redundante
vector_array_doble = ['lowlevel.gfcc.cov','lowlevel.gfcc.icov', 'lowlevel.mfcc.
    cov', 'lowlevel.mfcc.icov']
data = data.drop(vector_array_doble, axis=1)

for i in range(0,data.columns.size-2):
   columna = data.columns.values[i]
   #Se calcula la media de las columnas que son arrays para tener un solo valor
        por celda
   if data[columna].dtypes == "object":
        num = []
        for j in range(0,len(data[columna])):
            array = data[columna]
            #Para quitar los corchetes
            array_trans = array[j][1:-1]
            #Separa la cadena por las comas
            div_cadena = array_trans.split(",")
            #Se convierte en una lista de float
            floa = list(np.float_(div_cadena))
            #Calculo de la media de los elementos de la lista
            media_n = np.mean(floa, axis =0)
            #Se añade al Dataframe num
            num.append(media_n)


        new_num[str(columna)] = num
        data[columna] = num
        media_nm = new_num[columna].mean()
        desv_tipica_nm = new_num.loc[:,columna].std()

   #Calculo de la media de la columna
   media1 = data[columna].mean()
   #Desviación estándar de la columna
   desv_tipica = data.loc[:,columna].std()

   normalizacion = []
   for j in range(0,len(data[columna])):
        longitud = len(data[columna])
        normalizacion_Z= (data[columna][j]-media1)/desv_tipica
```

```
        normalizacion.append(normalizacion_Z)

   norm_df[str(columna)]=normalizacion

#Se incluyen las dos columnas que hemos convertido a binarias al nuevo
    Dataframe que contiene los valores ya normalizados
norm_df['tonal.chords_scale']= data['tonal.chords_scale']
norm_df['tonal.key_scale'] =data['tonal.key_scale']

#Busqueda de las columnas en blanco por ser todas las columnas iguales.
empty_cols = [col for col in norm_df.columns if norm_df[col].isnull().all()]

norm_df = norm_df.drop(empty_cols,
        axis=1)

print("Creando archivo csv...")

export_csv = norm_df.to_csv (r'/home/nieves/eclipse-workspace/prueba/prueba/
    data_normalized_v01.csv', index = None, header=True)

print("Archivo csv creado!")
```

3.2 Selección de los descriptores más importantes

Ya se han normalizado todos los datos y se poseen alrededor de 440 features con los que poder trabajar. Ante este gran número de características, es muy posible que algunas sean completamente irrelevantes o no sean importantes de cara a entrenar a un algoritmo para que sea capaz de predecir el palo al que corresponde cada archivo del corpus. También es importante tener en cuenta que todas estas características "inservibles" pueden provocar que la creación del modelo predictivo no sea eficiente, ya que se le van a destinar recursos que no deberían. Además, estas características pueden actuar como ruido, haciendo que el algoritmo funcione mal, precisando más tiempo y dando peores predicciones. Ante este problema, la solución más rápida y sencilla radica en el uso de un algoritmo de `feature_selection`.

`Feature_selection` es un proceso en el que se escogen las características más significativas de una base de datos dada. Merece la pena mencionar que esta técnica no es equivalente a una reducción de dimensiones al uso, ya que ésta última, al realizar esta reducción, crea nuevas combinaciones con los datos que se poseen, es decir, transforma las características; mientras que el método que se va a utilizar simplemente excluye elementos que no se consideran importantes. Para implementar todo lo mencionado se ha usado la librería de Python `scikit-learn`. La documentación referente a `feature_selection` se encuentra en [11].

3.2.1 Cómo funciona el algoritmo Feature_selection

Existen diferentes métodos presentes en la documentación de `scikit_learn` para aplicar las rutinas de `feature_selection`. En concreto, este trabajo se ha basado en el llamado *Univariate feature selection*, el cual se basa en el uso de diversas estadísticas para el cálculo de unas puntuaciones. Mientras la puntuación sea más alta, mayor será la relevancia que posea esa característica en COFLATUBE y, por tanto, más conviene que permanezca.

El método utilizado es el de `mutual_info_classif`, que se encarga de calcular la dependencia estadística entre dos variables. Será 0, si y solo si, las dos variables son independientes y tendrá valores elevados si sucede justo al contrario. Es así como la función se basa en estimar la entropía entre variables vecinas. Entre algunas de las variables a configurar es posible modificar el número de variables vecinas que usará para las estimaciones.

3.2.2 Aplicación del algoritmo

Como se ha visto en el capítulo anterior, existen una gran cantidad de palos presentes en COFLATUBE. Tanto es así, que antes de aplicar el algoritmo va a ser necesario hacer una reducción de estas categorías, ya que pueden causar problemas de overfitting, también llamado sobre-entrenamiento, que básicamente ocurre cuando no se le facilitan los datos para que pueda generalizar correctamente, sino que se le dan demasiadas opciones, haciendo que sea excesivamente específico. Para ello, se ha realizado un script para filtrar según los palos más frecuentes que se encuentran incluidos. En total son 12 palos diferentes: fandangos, soleás, bulerías, malagueñas, cantes de levante, granaínas, alegrías, cantiñas, caracoles, tangos, tientos, saetas y tonás.

Para realizar esta selección, se lleva a cabo un bucle que va a ir guardando los índices que tienen contenido en la columna `style` el nombre de alguno de los palos seleccionados. Estos índices se almacenarán en una lista auxiliar creada, que posteriormente se pasará como argumento a la función `iloc()` que seleccionará las filas y se almacenarán en un Dataframe diferente, el cual se exportará a un archivo csv. Tras esto, ha sido necesario hacer una revisión manual de los datos para eliminar ciertas filas, ya que existen filas que contienen varios palos a la vez. Esto se explica con que los archivos sonoros no dejan de ser actuaciones en directo donde unos cantes son precedidos por otros de otro palo sin pausa.

Código 3.2 subSelection_feature.py.

```
import pandas as pd

#Se crea un Dataframe con los valores normalizados
df2 = pd.read_csv('data_normalized_v01.csv')
data  = pd.DataFrame(df2)

#Se carga el archivo el archivo que contiene la BBDD de canciones como otro
    Dataframe
file = "videosFinalv1.xlsx";
filePath = "/home/nieves/eclipse-workspace/prueba/prueba/videosFinalv1.xlsx"
df1 = pd.read_excel(filePath);
enlaces = pd.DataFrame(df1);
long=len(enlaces.index);
col2 = enlaces['style'];

#Se añade la columna de palos la columna de palos
data['style']=col2

#lista de palos que se van a filtrar
lista_palos = ["fandango","sole","buler","malagueña","seguiriya","levante","
    granaína","alegr","cantiña",
              "caracole","tango", "tiento","saeta", "toná"]
long_palos = len(lista_palos)

#Se define una lista auxiliar para guardar los indices y el nuevo dataframe
df_aux=[]
df_selection = pd.DataFrame()

bandera =0
#En este bucle se guardan los índices que coinciden con la lista de palos
for i in range(0,len(col2)):
    bandera =0
    entra = 1
    for j in range(0,long_palos):
        if lista_palos[j] in col2[i].lower() and bandera ==0:
            bandera =1
            entra = 1+i
```

```
            df_aux.append(i)

#print(df_selection)
#En el dataframe de df_selection se guardan las filas con los indices guardados
     en la lista df_aux
df_selection = data.iloc[df_aux]

#print(df_selection.head(10))

print("Creando archivo csv...")

export_csv = df_selection.to_csv (r'/home/nieves/eclipse-workspace/prueba/
    prueba/sub_selection_styles.csv', index = None, header=True)

print("Archivo csv creado!")
```

Los resultados obtenidos se distribuyen de la siguiente forma:

Tabla 3.1 Canciones por palo.

Palos	Número
Fandangos	111
Soleares	157
Bulerías	314
Malagueñas	39
Seguiriyas	54
Cantes de Levante	51
Granaínas	18
Alegrías	99
Cantiñas	23
Caracoles	10
Tangos	115
Tientos	14
Saetas	37
Tonás	4
Total	1046

Tras ello, se necesita un último paso previo a la aplicación de `mutual_info_classif`, que consiste en convertir los palos seleccionados en números para que puedan introducirse en el algoritmo sin causar errores. Para realizar esto, se crea un diccionario donde se asignará un número a cada uno de los palos. Este diccionario se usará en un bucle que irá recorriendo la columna del Dataframe de 'styles', donde aparecen, y se irá consultando el diccionario en cada iteración para ver qué número asignarle a la celda en cuestión.

Código 3.3 mutual_info_classif_0.py.

```
import pandas as pd

#Se crea un Dataframe con el csv de la subseleccion
df2 = pd.read_csv('sub_selection_styles.csv')
data  = pd.DataFrame(df2)
palos_df = data['style']

#lista de palos que existen
lista_palos = ["fandango","sole","buler","malagueña","seguiriya","levante","
    granaína","alegr","cantiña",
              "caracole","tango", "tiento","saeta", "toná"]
longitud_palos = len(lista_palos)
```

```
#Se define el diccionario que se va a rellenar
new_dict = {}
lista=palos_df

#Se asigna un número a cada palo
for j in range(0, longitud_palos):
    new_dict[str(j)]= [m for m in palos_df if lista_palos[j] in m.lower()]

#Se recorre la columna y se va cambiando por el número correspondiente
for i in range(0, len(new_dict)):
    for j in range(0, len(palos_df)):
        if palos_df[j] in new_dict[str(i)]:
            lista[j]= i


#Se modifica la lista de string por la lista de números
data['style'] = lista


print("Creando archivo csv...")

export_csv = data.to_csv (r'/home/nieves/eclipse-workspace/prueba/prueba/
    sub_selection_styles_as_number.csv', index = None, header=True)

print("Archivo csv creado!")
```

La función `mutual_info_classif` tiene la siguiente forma:

```
sklearn.feature_selection.mutual_info_classif(X, y, *, discrete_features=áuto',
                n_neighbors=3, copy=True, random_state=None)
```

Donde `X` es el array o matriz con todas las características, a excepción de la columna objetivo, la cual es `Y` y que en el caso que ocupa este trabajo sería la columna de palos o 'styles'. El resto de parámetros se han mantenido con sus valores por defecto, como se indica en la tabla 3.2.

Tabla 3.2 Parámetros de la función de Mutual Information.

Parámetros	Descripción	Valores
discrete_features	Forma de considerar las variables	auto
n_neighbors	Nº de vecinos para usar en la estimación de Mutual Information	int:3
copy	Si se hace una copia de los datos dados	True
random_state	Creación de variables aleatorias para eliminar valores repetidos	None

La salida del algoritmo consistirá en una lista con las puntuaciones que posee cada una de las características existentes. Mientras más altos sean esos valores, mayor será la relevancia de ese feature en COFLATUBE y más importante, por ello, la necesidad de conservarlo. Tras obtener estos valores, se realiza una criba para eliminar aquellas características por debajo del valor que se estima. En este caso se han eliminado todas aquellas que poseen una puntuación menor de 0.25 y se guardan en un nuevo archivo, listas para el siguiente paso.

Código 3.4 mutual_info_classic_v01.py.

```
from sklearn.feature_selection import mutual_info_classif
import pandas as pd
```

```
#Se carga el archivo en un dataframe
df2 = pd.read_csv('sub_selection_styles_as_number.csv')
data  = pd.DataFrame(df2)

#se indica cual es la X y cual la Y
X = data.drop('style', axis=1)
Y=data['style']

#se aplica el algoritmo de scikit-learn
list_scores=mutual_info_classif(X,Y)

# Se eliminan las caracteristicas cuya puntuacion es inferior a 0.25
list_names =[]
array_delete_features=[]
for i in range(len(list_scores)):
    columna = data.columns
    list_names.append(columna[i])
    #print( str(columna[i]) + ' Score: '+ str(list_scores[i]))
    if list_scores[i] <0.25:
        array_delete_features.append(list_names[i])

print(array_delete_features)

#Borramos columnas que son menores a la puntuacion indicada, es decir las que
    se han guardado en la lista
data = data.drop(array_delete_features, axis=1)

print("Creando archivo csv...")

export_csv = data.to_csv (r'/home/nieves/eclipse-workspace/prueba/prueba/
    mutual_info_classif_27 features.csv', index = None, header=True)

print(" Archivo csv creado!")
```

3.3 Reducción de las dimensiones de los datos

Una vez se disponen de los features mas representativos, ya es posible aplicar el último de los algoritmos, el llamado Manifold Learning, el cual está enfocado en la reducción de la dimensionalidad de forma no lineal. La necesidad de reducir las dimensiones de las bases de datos viene por la existencia de diversas dificultades para poder visualizar los resultados. En este trabajo se disponen de 27 features, lo que se traduce a 27 dimensiones: estos valores generarían unos resultados poco intuitivos, además de muy costosos computacionalmente, por lo que se aplicará el algoritmo `Manifold` con el objetivo de obtener únicamente 2 y así representar en el eje X e Y. La forma en la que el algoritmo realiza esta reducción se basa en definición de rasgos específicos de los datos para realizar una proyección lineal de los datos, aunque depende en gran medida del método que se utilice. El que se va a usar en este caso es `Multi-dimensional Scaling(MDS)`.

3.3.1 Cómo funciona el algoritmo de Multi-dimensional Scaling

`Multi-dimensional Scaling(MDS)` busca la reducción de la dimensionalidad mediante la búsqueda de un conjunto de vectores en el espacio n-dimensional, de tal forma que la matriz de distancias euclídeas sea la mínima posible entre ellos y la matriz de entrada, dando como salida una variable llamada "stress" que no es más que la suma de las distancias al cuadrado de las disparidades y las distancias para todos los puntos. Los datos que son más similares serán los más próximos, mientras que sucede justo lo contrario para los datos que resultan ser más diferentes.

La clase viene definida por:

```
class sklearn.manifold.MDS(n_components=2, *, metric=True, n_init=4,
max_iter=300, verbose=0, eps=0.001, n_jobs=None, random_state=None,
                    dissimilarity="euclidean")
```

Donde los parámetros que se han utilizado son:

- **n_component**: Es el número de dimensiones que se quieren obtener como resultado final.
- **metric**: Existen dos tipos de algortitmo MDS en función de este parámetro. Si es `True`, se usará un método para la reducción de la variable "stress", es decir, minimizando la suma de cuadrados de disparidades y distancias. Por el contrario, si se selecciona `False`, se opta por el método no métrico, el cual intenta disminuir el valor de 'stress'mediante cálculos de regresión que intenten preservar el orden de las distancias buscando una relación entre las distancias, en el espacio y las disparidades. En este caso, se opta por la elección de `True` ya que, por lo general, se obtienen mejores resultados.
- **max_iter**: Número de iteraciones que realizará el algoritmo, por defecto son 300. Para mejorar los resultados, se ha modificado para que sea de 1000 iteraciones y se pueda afinar más el resultado.

El resto de parámetros mantiene sus valores por defecto, como se detalla en la tabla 3.3.

Tabla 3.3 Parámetros de la función de MDS.

Parámetros	Descripción	Valores
n_init	Número de veces que el algoritmo se ejecutará con diferentes inicializaciones	4
verbose	Nivel de detalle	0
eps	Tolerancia	10^{-3}
n_jobs	Número de procesadores	None = 1
random_state	Determina si hay generador de números aleatorios	None
dissimilarity	Medida de la dimisilaridad	"euclidean"

3.3.2 Aplicación del algoritmo

Para implementar el script se ha creado una función llamada `apply_manifold_learning(X, method, lista_palos)` con tres parámetros: X, la matriz con los valores a entrenar, el método a usar (que en este caso será MDS), y por último la lista de palos, que será necesaria para la visualización de los resultados de una manera más clara y legible. Una vez se llama a la función descrita, se llama al método `fit()` para entrenar o aplicar el algoritmo MDS. Una vez se ha realizado este paso, se pasan los valores generados a un DataFrame, esto permitirá manipular los datos de una forma sencilla.

Para la representación gráfica se ha usado la librería `matplotlib`, ya mencionada en anteriores capítulos, la cual posee una infinidad de métodos para visualizar los datos. En este caso se ha usado `pyplot`, que provee un framework similar al que posee Matlab. Junto con la librería numpy se engloban en pylab, el cual es muy útil para los trabajos interactivos.

Para representar si los algoritmos utilizados han clasificado por palos correctamente, se hace necesario definir una paleta de colores. Esto se puede hacer de forma automática, usando la librería `seaborn`, o bien manualmente, mediante una lista que contenga los nombres de los colores. Lógicamente, la mejor opción sería usar la generación automática de colores; sin embargo, el número de colores son limitados en cada paleta y a menudo aparecen colores de tonalidades y saturación parecidas, por lo que hace complicada su diferenciación. Es por esta razón por la que se ha optado por crear una lista de colores a mano con colores especialmente seleccionados. Además de los colores, se ha creado una lista de strings con el nombre de los palos que aparecerán en la leyenda de la gráfica, ya que hasta ahora los palos estaban identificados por números.

Para crear la gráfica y configurar el tamaño de la misma, los ejes, etc se utiliza la función de matplotlib `figure()` y, para incluir los datos, la función `scatter()`, a la cual se le han añadido varios parámetros importantes: En primer, lugar las coordenadas, `X(0)` y `X(1)`, que corresponden a la coordenada x e y respectivamente; seguidamente, el mapa de colores `cmap`, que se introduce como lista con la variable creada manualmente `color_p`. Para la visualización de la leyenda, se ha creado una variable llamada `handleList`,

que sencillamente indica que un marcador de forma circular tendrá el color que le corresponde en función del palo que representa. Esta variable se añadirá como parámetro a la función `legend()`.

El código completo del uso del algortimo manifold y la representación de los datos es el siguiente:

Código 3.5 manifold.py.

```
from sklearn.manifold import MDS
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
import seaborn as sns

#Se cargan los datos en un Dataframe
df2 = pd.read_csv('mutual_info_classif_27 features.csv')
data  = pd.DataFrame(df2)


X = data.drop('style', axis=1)
Y=data['style']

#La lista de palos para la leyenda
lista_palos = ["Fandangos","Soleás","Bulerías","Malagueñas","Seguiriyas","
    Cantes de Levante",
                "Granaínas","Alegrías","Cantiñas",
            "Caracoles","Tangos", "Tientos","Saetas", "Tonás"]
tam=len(lista_palos)

#Para crear colores de forma automatica
pal1 = sns.color_palette("Set2")
pal2 = sns.color_palette("Paired", tam-len(pal1)+1)
palette =pal1 + pal2

#Paleta creada manualmente
color_p=['red', 'blue','darkorange','gold','darkolivegreen','springgreen', '
    purple', 'navy','magenta','pink',
       'darkgrey','mediumaquamarine','lightcoral','brown']

#función para aplicar algoritmo manifold y representar
def apply_manifold_learning(X, method, lista_palos):
    #Se entrena el algoritmo con las variables X
    X=method.fit_transform(X)

    print("New shape of X : " , X.shape)
    print()
    print("Sample X : \n")

    X=pd.DataFrame(X)
    print(X.sample(10))


    fig = plt.figure(figsize=(10,8))
    plt.scatter(X[0], X[1], c=Y, cmap=matplotlib.colors.ListedColormap(color_p)
        )

    #Asignar un color diferente a cada palo en la leyenda
```

```
    handlelist = [plt.plot([],marker='o', color=color)[0] for color in color_p]

    #Se crea una leyenda que asigne al numero Y el palo que le corresponda
    plt.legend(handlelist, lista_palos,bbox_to_anchor=(0.75, 0.5), title = "
        Palos" )
    plt.title('Clasificación palos mediante algoritmo MDS')
    plt.xlabel('Componente 1')
    plt.ylabel('Componente 2')
    plt.show()


    return method

#Llamada a la función
mds = apply_manifold_learning(X, MDS(n_components = 2, metric = True, max_iter
    =1000), lista_palos)
```

4 Análisis de los datos y conclusiones

Una vez finalizadas las implementaciones de código, ya es posible sacar ciertas conclusiones acerca del potencial que tiene este método en la discriminación de los palos.

4.1 Análisis de las imágenes obtenidas

El resultado obtenido en la ejecución del código 3.5 se refleja en la figura 4.1. A continuación se van a analizar y se profundizará en los resultados obtenidos, además de sacar algunas conclusiones sobre su uso en la clasificación de palos de flamenco.

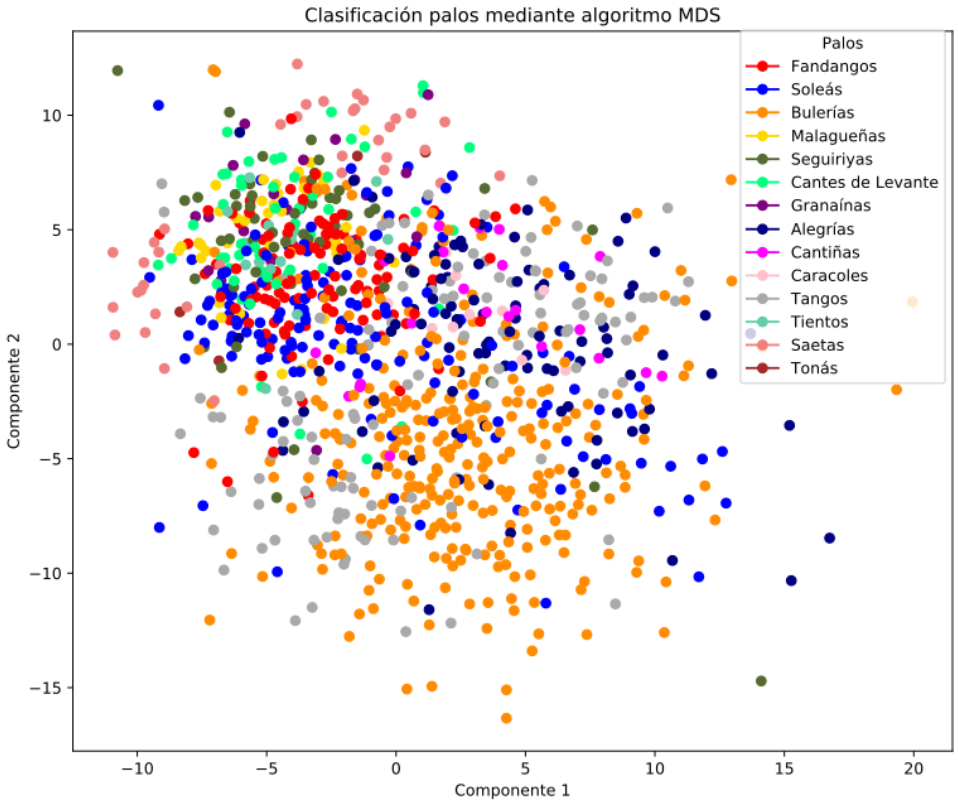

Figura 4.1 Clasificación de los palos mediante algoritmo MDS.

En primer lugar, puede verse cómo los cantes más festivos, como las bulerías, las alegrías o los tangos se encuentran cercanos en la gráfica, mientras que, por el contrario, los fandangos , malagueñas y seguiriyas se mantienen agrupadas y alejadas de este primer grupo, ya que su ritmo y tonalidad son totalmente diferentes. Esto hace indicar que la clasificación mediante descriptores con el algoritmo MDS es acertada.

Por otra parte, de cara a tener una visión más específica, se han llevado a cabo subselecciones de los palos en grupos de 2 y 3. A continuación se explican los resultados obtenidos, que se han ido comparando con [8]:

- **Bulerías y Soleás**: Figura 4.2. La representación entre las bulerías y las soleás sugiere que existen diferencias entre ellas a pesar de tener el mismo compás. Por una parte, las soleás poseen un tempo lento y sobrio; por la otra, las bulerías son uno de los palos 'fiesteros'por antonomasia. Dado que las 'bulerías por soleá'poseen, en su mayoría, la melodía de las soleás y el ritmo de las bulerías, se explica que no exista una separación clara entre los dos grupos.

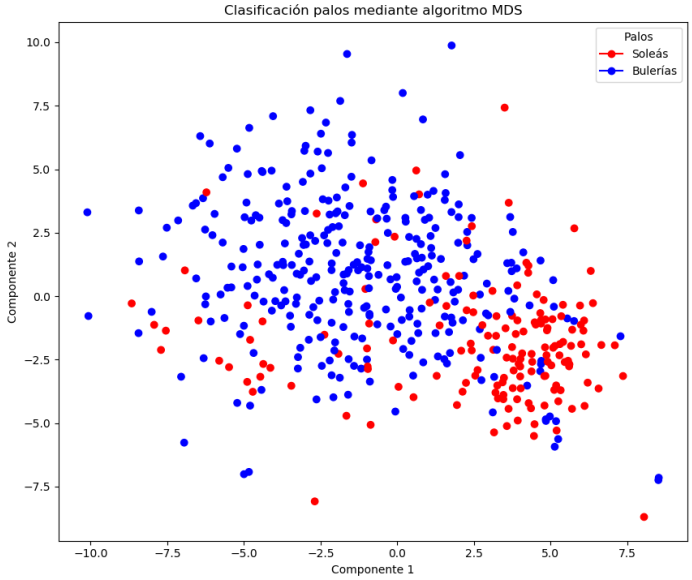

Figura 4.2 Bulerías y Soleás.

- **Bulerías, Alegrías y Tientos**: En estos tres palos se producen similitudes dos a dos, es decir, por una parte las bulerías y las alegrías poseen el mismo ritmo y por otra, las bulerías y los tientos se mantienen en la misma tonalidad a pesar de tener ritmos distintos. Esta dualidad puede provocar, tal y como se muestra en la figura 4.3, que los palos no aparezcan bien diferenciados haciendo parecer que la clasificación es enteramente errónea.

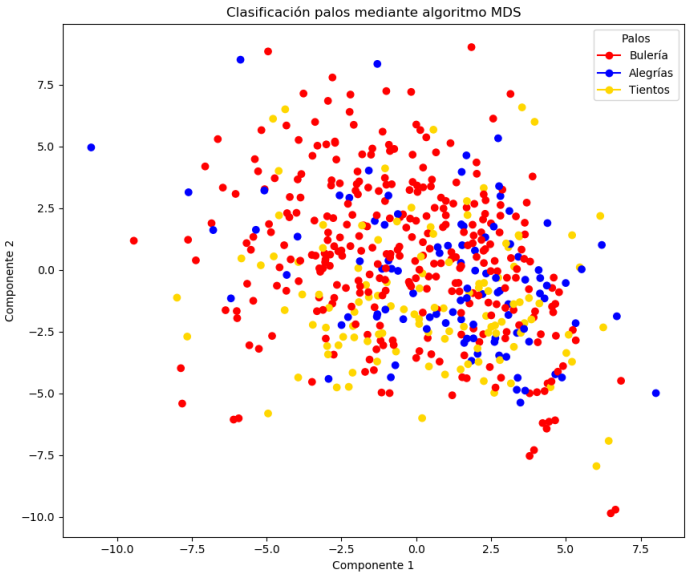

Figura 4.3 Bulerías, Alegrías y Tientos.

- **Bulerías y Fandangos**: Entre ambos palos existen importantes diferencias que radican en la tonalidad y la métrica. Por ello, es posible observar, en la figura 4.4, una clara discriminación de los palos representados.

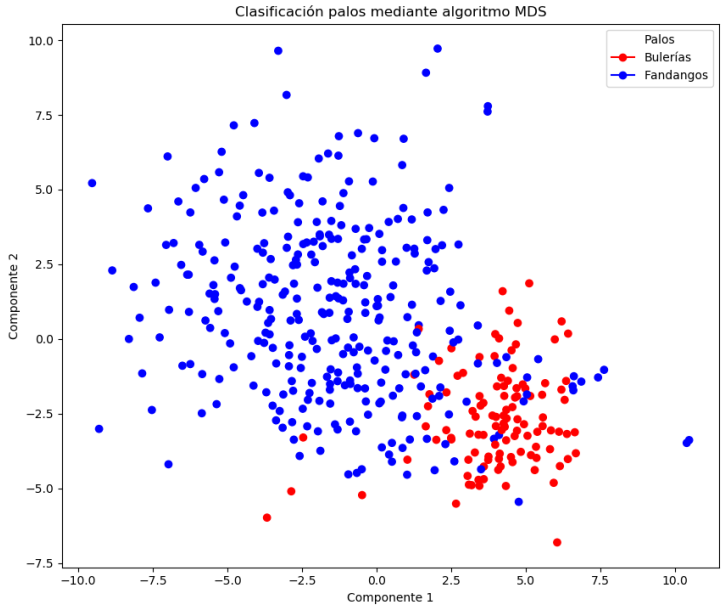

Figura 4.4 Bulerías y Fandangos.

- **Caracoles, Tangos y Tientos**: Figura 4.5. Los tangos se diferencian de los tientos por ser algo más rápidos, por lo que aparecen bien diferenciados. Los caracoles, por su parte, pueden observarse mezclados con los tangos. Esto puede explicarse por el hecho de que poseen el mismo tempo.

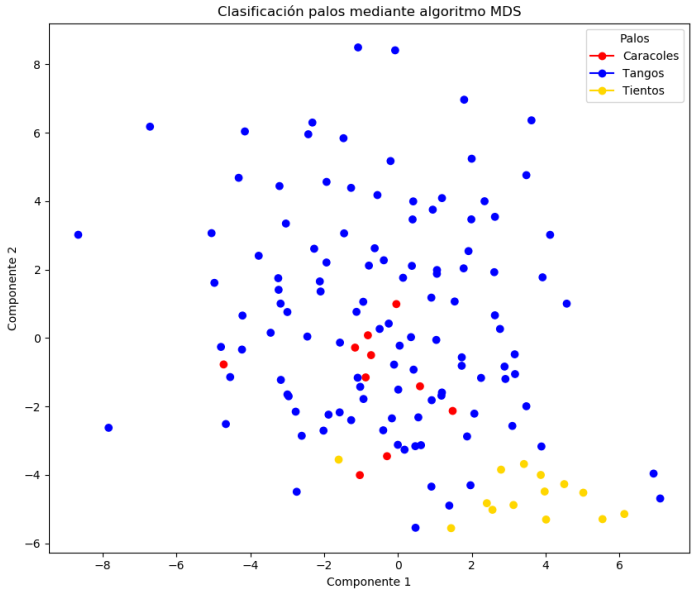

Figura 4.5 Caracoles, Tangos y Tientos.

- **Fadangos y Seguiriyas**: Observando la figura 4.6 puede intuirse la existencia de cierta similitud entre los palos debido a su cercanía, sin embargo, los fandangos y las seguiriyas son totalmente diferentes, poseen distinto ritmo y tempo. La justificación a este resultado radica en el hecho de que los fandangos existentes en COFLATUBE sean fandangos naturales (aparecen con gran frecuencia en los archivos de RTVA), los cuales si que guardan similitudes con las seguiriyas. Los fandangos de Huelva, podrían ser aquellos puntos que aparecen más alejados.

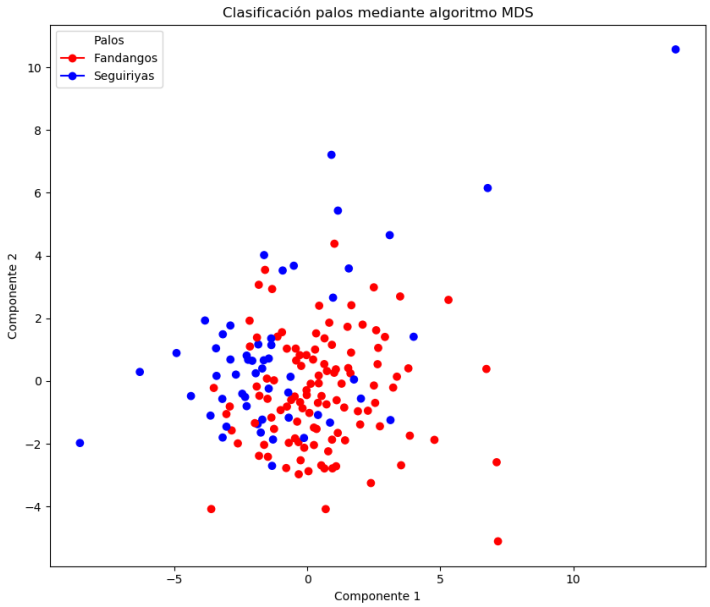

Figura 4.6 Fadangos y Seguiriyas.

- **Fandangos, Malagueñas y Cantes de Levante**: En la figura 4.7 puede observarse una diferenciación entre fandangos y malagueñas. Esto puede explicarse por el hecho de que los fandangos están cantados con un cierto compás, mientras que, por el contrario, las malagueñas son mucho más lentas. Por otra parte, los cantes de Levante, los cuales son cantes derivados de las malagueñas, poseen un abanico más amplio en cuanto a tempos y compás se refiere. Por esta razón, los cantes de Levante aparecen mezclados tanto con los fandangos como con las malagueñas.

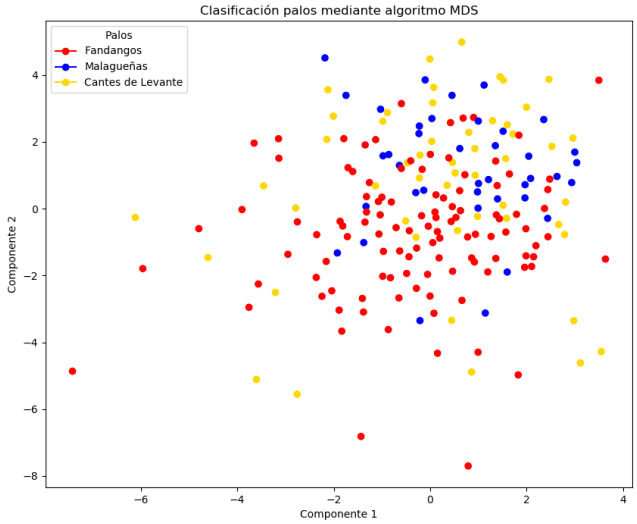

Figura 4.7 Fandangos, Malagueñas y Cantes de Levante.

- **Granaínas y Saetas**: Las granaínas son un subpalo de los fandangos con una métrica marcada y una tonalidad bimodal, mientras que las saetas presentan un compás libre propio de los cantes a cappella, donde el cante es el motor de toda la canción sin apenas presencia de otros elementos, por lo que la diferencia de ambos palos que se observa en la figura 4.8 está justificada.

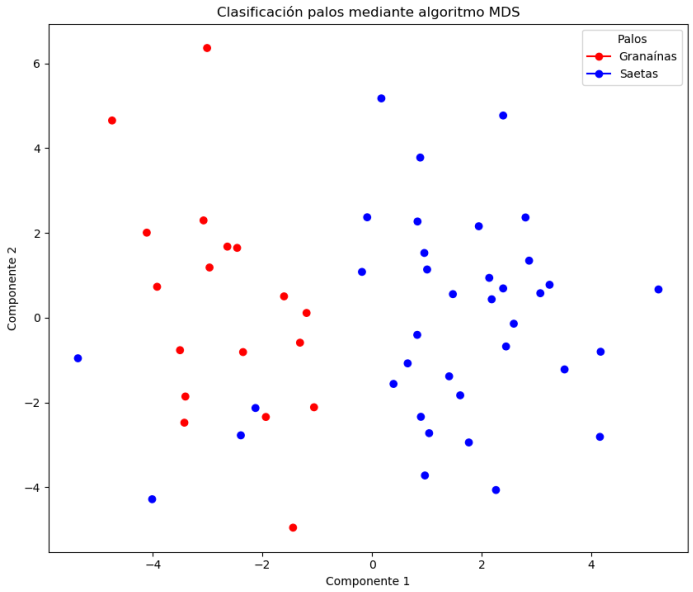

Figura 4.8 Granaínas y Saetas.

4.2 Conclusiones

El incremento de la popularidad del flamenco en todo el mundo es ya más que evidente gracias, entre otras razones, a su digitalización, que lo hace mucho más accesible a todos los públicos mediante plataformas online como Spotify o YouTube.

El estudio computacional del flamenco es, hoy día, un terreno de investigación en aumento donde, en general, el uso de herramientas como las descritas en este trabajo para la descripción y análisis del flamenco son realmente interesantes y prometen arrojar resultados aún más prometedores de los ya existentes en la actualidad. Por ello, esta puede ser una de líneas a seguir desarrollando en el futuro.

Este trabajo contiene el proceso de desarrollo y análisis mediante la minería de datos y la aplicación de un proceso de selección y discriminación de características de tipo estadístico. Estos métodos han sido aplicados a un gran corpus de grabaciones comerciales de flamenco. El objetivo consistía en la realización de un estudio de similitud de una base de datos llamada COFLATUBE, procedente de un ente público como es RTVA, evitando el uso de métodos manuales, los cuales requerirían una gran cantidad de tiempo. Además, este estudio permitió descubrir el potencial de este tipo de datos o anotaciones para la discriminación y clasificación de los palos del flamenco.

Los resultados obtenidos permiten verificar que los parámetros rítmicos y tonales generados por el algoritmo *Essentia* no son suficientes para distinguir palos, ya que las emociones que se transmiten en los cantes juegan, también, un papel fundamental. Esto puede comprobarse en la figura 4.1 si se comparan, por ejemplo, las bulerías y las alegrías o los fandangos y los cantes de levante.

Por último, y de cara al futuro, sería de interés investigar qué otros parámetros objetivos podrían permitir una discriminación más precisa de los palos. Otra tarea realmente importante es la de crear nuevos corpus más extensos y diversos que aumenten la capacidad de estudio y análisis para los algoritmos de selección de características, ya que, desafortunadamente, en el flamenco es difícil encontrar grandes colecciones de archivos a las que acceder, algo que no ocurre, por ejemplo, con géneros más extendidos como el pop. Todo lo mencionado ayudaría a comprender de forma más analítica el flamenco, una música de orígenes tan diversos, de una naturaleza tan subjetiva y de unos límites aún por descubrir.

Este trabajo, comprendido dentro del proyecto COFLA, ha generado una base de datos completamente nueva con la intención de ser de utilidad para futuros trabajos que lleven a cabo aplicaciones que usen aprendizaje profundo y redes neuronales en la tarea de clasificación de estilos dentro del flamenco.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Hervé Abdi, *Normalizing Data*, In Neil Salkind (Ed.), Encyclopedia of Research Design (2010).

[2] Dmitry Bogdanov, Nicolas Wack, Emilia Gómez, Sankalp Gulati, Perfecto Herrera, O. Mayor, Gerard Roma, Justin Salamon, J. R. Zapata, and Xavier Serra, *Essentia: an audio analysis library for music information retrieval*, International Society for Music Information Retrieval Conference (ISMIR'13) (Curitiba, Brazil), 04/11/2013 2013, pp. 493–498.

[3] E. Gómez y J. Mora F. Gomez, J.M. Díaz-Bañez, *Flamenco music and its Computational Study.*, In BRIDGES: Mathematical Connections in Art, Music, and Science (2014), 119.

[4] Fabrice Bellard, *https://www.ffmpeg.org/ffmpeg.html, Documentación FFmpeg*, 2000.

[5] Faustino Núñez, *http://www.flamencopolis.com/, Flamencópolis*, 2011.

[6] W. A. Schloss M. Wright G. Tzanetakis, A. Kapur, *Computational Ethnomusicology and Music Information Retrieval*, Department of Computer Science and School of Music, University of Victoria, Canada.

[7] Díaz-Báñez J.-M. Mora J. Kroher, N. and E. Gómez, *Corpus COFLA: a research corpus for the computational study of flamenco music.*, Journal on Computing and Cultural Heritage (JOCCH) (2016), 9.

[8] Nadine Kroher, *Flamenco Music Information Retrieval. Automatic Content-Based Description of Flamenco Music Collections. (English)*, (2018), 164.

[9] Joaquín Mora Emilia Gómez Nadine Kroher, Jose Miguel Díaz Báñez, *Corpus COFLA: A research corpus for the computational study of flamenco music. (English)*, ACM J. Comput. Cult. Herit. 0, 0, Article 0 (2015).

[10] Sergio Oramas and Mohamed Sordo, *Knowledge is out there:a new step in the evolution of music digital libraries*, Fontes Artis Musicae, Volume 63/4 (2016).

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research **12** (2011), 2825–2830.

[12] Python Standard Library, *https://docs.python.org/3/library/subprocess.html, Documentación subprocess*.

[13] Ricardo García, *https://github.com/ytdl-org/youtube-dl, Documentación Youtube-dl*, 2006.

[14] Rob Van Bekkum, Jorden Van Breemen, Steven Gosseling, Sander Van der Oever, *https://delftswa.gitbooks.io/desosa2016/content/youtube-dl/chapter.html, Easy understanding of youtube-dl*, 2016.

[15] Perfecto Herrera Boyer y Emilia Gómez Gutiérrez, *Tecnologías para el análisis del contenido musical de archivos sonoros y para la generación de nuevos metadatos*, Music Technology Group, Departamento de Tecnologías de la Información y las Comunicaciones, Universitat Pompeu Fabra y Departamento de Sonología, Escola Superior de Música de Catalunya.