Sistema de clasificación de géneros musicales

Darío Fernando Soto - 0815067
dafsotomo@unal.edu.co
Andrés Camilo Salazar - 0815546
acsalazar@unal.edu.co

Profesor: Andrés Marino Álvarez Meza Ph. D.
Teoría de Señales 2019-I
Departamento de ingenieria Electrica, Electronica y Computacion
Universidad Nacional de Colómbia - sede Manizales
22 de Julio - 2019

I. Introducción

En este trabajo se lleva a cabo la creación a detalle y el análisis de un sistema de clasificación para géneros musicales utilizando técnicas de aprendizaje de maquina. Creamos una base de datos de 160 canciones clasificadas en 4 géneros: Electrónica, Rap, Rock y Salsa. Esta base de datos se encuentra constituida por: las canciones en formato *.mp3*; luego, mediante el desarrollo del código, se convierten a formato *.wav*; y finalmente, se crea una matriz $\mathbf{X}_{ij}$ donde i corresponde al numero de canciones de la base de datos y j corresponde al numero de datos que contiene cada canción.

Posteriormente, se utiliza el algoritmo *Random Forest* también conocido en castellano como *Bosques Aleatorios*, el cual, es una combinación de árboles predictores, tal que cada árbol depende de un vector aleatorio creado a partir de los datos de entrenamiento y probado con los datos de prueba.

Implementamos un sistema de modulación por amplitud utilizando la técnica de *Double sideband with suppried carrier* para transmitir la canción digitalmente y posteriormente demodularla con la técnica *Synchronous detection* para que pueda ser clasificada por el algoritmo.

Todos los códigos implementados en este proyecto, se desarrollan en el lenguaje de programación **Python** y en el ambiente de programación *Jupyter* que se encuentra dentro del paquete *Anaconda Navigator*^{R}.

II. Objetivo General

Implementar modulación y demodulación digital para una señal de audio de entrada; y un sistema de clasificación en genero musical para la misma, utilizando técnicas de modulación por doble banda lateral con portadora suprimida, demodulación por detección síncrona, aprendizaje de maquina y programación general.

III. Objetivos Específicos

- Construir una base de datos de 40 segmentos de canciones de 5 segundos clasificadas en 4 géneros musicales.
- Implementar un sistema de modulación por doble banda lateral con portadora suprimida y demodulación por detección síncrona para la transmisión y recepción de la señal
- Implementar el algoritmo de aprendizaje de maquina *Random Forest* para la clasificación de la señal de entrada en uno de los cuatro géneros musicales que se encuentran presentes en la base de datos.

IV. Estado del Arte

Actualmente existen muchos sistemas de reconocimiento de géneros musicales en el mercado, llegando hasta tal punto que todas las personas con un teléfono celular podemos cargar uno en nuestro propio bolsillo. Esto es debido a que los grandes sistemas operativos actuales realizan esta apuesta, yendo incluso un paso mas adelante, ya que a parte de reconocer el genero musical, también son capaces de reconocer la canción, el artista, el álbum y el año en la cual fue lanzada, además de proporcionar un link para escucharla en una de las plataformas musicales como: Spotify, Dezzer, Youtube, etc.

Los sistemas de reconocimiento musical mas conocidos en la actualidad, se encuentran integrados dentro de los asistentes personales que tiene el sistema operativo de cada celular, por ejemplo: Siri de *Apple*, Cortana de *Microsoft*, Alexa de *Amazon* y el asistente de *Google*. De esta forma, reconocer un genero musical o una canción es tan simple como preguntarle al asistente "¿Qué canción esta sonando?", para que este active el micrófono, grabe 10 segundos de la canción y proporcione toda la información de la misma. Aparte de esto asistentes personales, existen otras aplicaciones para celulares y computadores que cuentan con este mismo sistema como lo es *Shazam*.

La empresa de desarrollo tecnológico *Mobile technologies*, fue la pionera en estos algoritmos de reconocimiento musical con su aplicación *Shazam* lanzada en el año 2003 y creada por *Avery Li-Chung Wang*. Este algoritmo funciona mediante la creación de una huella digital tambien conocida como *Espectrograma*.

La canción que se desea clasificar, es grabada mediante el micrófono del celular, el algoritmo inicia con un filtro digital para omitir el ruido del ambiente y guardar únicamente la información de la canción. Las señales de amplitud eléctrica en el dominio del tiempo, son transformadas al dominio de la frecuencia, utilizando el algoritmo *Discret Fourier Transform (DFT)* y teniendo la señal en el tiempo y su espectro de frecuencias, se procede a construir un espectrograma, el cual corresponde a la huella digital de la canción. Como se observa en la Figura 1. en el eje X tenemos el tiempo transcurrido y en el eje Y tenemos las frecuencias para cada instante de tiempo, ademas, encontramos una tercera dimension, la cual es el color, que representa los pesos para cada una de las frecuencias que componen la señal de sonido en cada instante de tiempo.

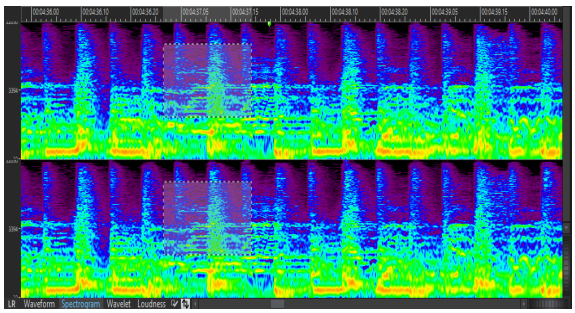

Figura 1. Espectrograma.

Luego de tener el espectrograma de la señal grabada se procede a comparar la entrada con una base de datos de mas de 1 millón de canciones, por lo tanto finalmente, el algoritmo de Shazam posee un sistema aproximación debido al error de la grabación y posteriormente un sistema de clasificación por distancias, para determinar la huella digital con mayor similitud a la señal grabada.

V. Desarrollo y Análisis del sistema

Para el desarrollo de nuestro sistema de clasificación de géneros musicales, optamos por el lenguaje de programación *Python* y trabajamos en el entorno *Jupyter* dentro de el paquete de programación de *Anaconda Navigator*. En la Figura 2. se muestran todas las librerías utilizadas para el desarrollo del código.

```
import os                                          # Interact with the operating system
import errno                                       # Standard errno system symbols
import csv                                         # Interact with .csv files
import soundfile as sf                             # Process audio files
import numpy     as np                             # Mathemtical functions
import pandas    as pd                             # process the data
import seaborn   as sns                            # Statistical data visualization
from sklearn.model_selection import train_test_split  # Split the data(training,test)
from sklearn.preprocessing   import StandardScaler    # Feature Scaling
from sklearn.ensemble        import RandomForestClassifier # Train Algoritm
from sklearn.metrics         import confusion_matrix  # Measuring of algorith perfomance
from sklearn.metrics         import classification_report # Report of the algorithm accuracy
from sklearn.externals       import joblib         # Save Model
from matplotlib              import pyplot as plt  #Plot
plt.style.use('ggplot')
```

Figura 2. Librerías para la construcción del sistema

- *Os*: Interactuar con el sistema operativo y ejecutar comandos de CMD en Python.
- *errno*: Errores estándar proporcionados por el sistema *Windows 10.*
- *csv*: Interactuar con archivos que tengan extensión *.csv*
- *Soundfile*: Procesamiento y operación de archivos de audio en Python.
- *numpy*: Realizar todo tipo de operaciones y funciones en el campo de las Matemáticas.
- *pandas*:Almacenar la base de datos en el disco duro y procesarla para su análisis.
- *seaborn*: Visualización de datos estadísticos para la etapa de clasificación.
- *sklearn*: Librería que proporciona todos los algoritmos necesarios para implementar técnicas de aprendizaje de maquina.
- *matplotlib*: Realizar gráficas de los datos para facilitar la comprensión de los mismos.

V-A. *Base de Datos*

La elaboración de una buena base de datos es el factor fundamental para el éxito o fracaso de nuestro proyecto. El propósito es realizar la clasificación de canciones dentro de cuatro géneros. Escogimos: Electrónica, Rap, Rock y Salsa; debido a que entendemos que en las técnicas y algoritmos de aprendizaje de maquina, existen ciertos aspectos que el computador nunca va a ser capaz de desarrollar pues no pose la capacidad de comprensión de datos que tenemos los seres humanos.

Por esta razón, es nuestro deber orientar el aprendizaje de maquina hacia el éxito del problema, para ello, los cuatro géneros musicales deben poseer características completamente distintas tanto en el dominio del tiempo como en la frecuencia. Esto es, la representación espectral de las frecuencias que conforman una canción del genero *salsa*, que son básicamente instrumentos de viento, es muy distinta a la representación espectral de el genero *Electrónica*, creada en su mayoría por sonidos generados en computadora.

En el momento de abordar este problema, es preciso tener claro que la diferencia entre el **color** de un instrumento y otro a pesar de que estén tocando las mismas notas, es su representación espectral, por lo tanto, mientras que en Salsa tenemos instrumentos de viento, en Rock tenemos guitarras con mucha distorsión y en Rap, tenemos mucha voz con melodías simples, por ultimo, en Electrónica tenemos sonidos de sintetizadores con poca voz. Además, es necesario manejar, la misma cantidad de datos para cada canción, por esto, en la Figura 3. se muestra las especificaciones con las que se trabajara en todo el código.

```
genre   = ['electro','rap','rock','salsa']  # genre's vector
fs      = 44100                             # Standard sample frequency for all songs
[ti,tf] = [40,45]                           # Standard Time segment to trim the songs
```

Figura 3. Características generales para el desarrollo del código

V-A1. Construcción: En este apartad, empezamos con la selección de las canciones como se explico anteriormente. Se seleccionaron 160 canciones dividas en los 4 géneros (40 canciones para cada genero). Las canciones se descargaron desde la plataforma musical *Youtube*, debido a que buscamos igual calidad de sonido en toda la base de datos[1].

Para facilitar el proceso de descarga de las canciones, estas se realizan en *.mp3* que es un formato de audio de datos comprimidos. Estos datos no pueden ser procesados mediante una herramienta computacional, razón por la cual toda la base de datos debe estar en formato *.wav*. En la Figura 4. se muestra el proceso de conversión de todas las canciones de la base de datos a archivos con extensión .wav

```
for i in range(len(genre)):
    try:
        os.mkdir('Database/' + genre[i] + '/wavfiles')     # Make wavfiles folder
    except OSError as e:
        if e.errno != errno.EEXIST:                        # System error
            raise
    for j in range(40):
        try:
            n_in  = 'Database/' + genre[i] + '/Track_' + str(j+1) + '.mp3'
            n_out = 'Database/' + genre[i] + '/wavfiles/Track_' + str(j+1) + '.wav'
            os.system('ffmpeg -i ' + n_in +' '+ n_out)     # Convert .mp3 files to .wav files
        except OSError as e:
            if e.errno != errno.EEXIST:
                raise
```

Figura 4. Conversión de formato *.mp3* a *.wav*

Una vez tenemos nuestra base de datos construida en formato *.wav* podemos analizar los datos de las canciones desde el código de python, para ello utilizamos la librería **Soundfile**, la cual proporciona algoritmos que nos facilitan el procesamiento de los archivos de audio.

Para la construcción de la matriz que conforma la base de datos, es necesario recortar cada canción en un segmento de 5 segundos (este parámetro se muestra en la Figura 3.) debido a que queremos estandarizar y disminuir el tamaño de los datos de cada canción para que no suponga un gran esfuerzo computacional a la hora de procesarlos y que no ocupe un tamaño considerable en el disco duro. la construcción de la matriz base de datos en el dominio de el tiempo, se muestra en la Figura 5.

Los archivos de sonido en formato estéreo cuentan con información de ambos canales, lo cual se representa en una matriz X_{ij} donde i representa el numero de datos de la canción y $j = 2$ canales de información. Si utilizamos los datos de esta forma, seria necesario implementar una matriz de 3 dimensiones, lo cual dificultaría, el desarrollo del algoritmo de clasificación, por esta razón, decidimos concatenar los dos canales en un solo vector unidimensional para armar la matriz base de datos.

[1]El principal parámetro para la calidad de sonido es la frecuencia de muestreo $f_s = 44100\ Hz$

```
# Make the Database matrix
Database=np.zeros([160,10*fs])
for i in range(0,len(genre),1):
    for j in range(40):
        if (i==0): k=j      # Genre[0] = Electro
        if (i==1): k=j+40   # Genre[1] = Rap
        if (i==2): k=j+80   # Genre[2] = Rock
        if (i==3): k=j+120  # Genre[3] = Salsa
        x,a=sf.read('Database/'+genre[i]+'/wavfiles/Track_'+str(j+1)+'.wav')
        xdati=x[int(ti*fs):int(tf*fs),0]
        xdatd=x[int(ti*fs):int(tf*fs),1]
        Database[k,:]=np.concatenate((xdati,xdatd),axis=None)
```

Figura 5. Construcción de la base de datos en el dominio del tiempo

V-A2. Transformada rápida de Fourier: La Transformada rápida de Fourier, conocida por la abreviatura FFT (del inglés **Fast Fourier Transform**) es un algoritmo eficiente que permite calcular la transformada de Fourier discreta (DFT) y su inversa.

La Transformada rápida de Fourier es un algoritmo computacional que se basa en la **Transformada discreta de Fourier DFT**.

Sea $x[n]$ una señal aperiódica discreta en el tiempo. La transformada discreta de Fourier (DFT) de esta señal se define como:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j2\pi k \frac{n}{N}} \quad (1)$$

$$k = 0, 1, 2, ..., N-1$$

en la cual $X[k]$ es un conjunto de números complejos. La evaluación directa de esa fórmula requiere N^2 operaciones aritméticas, pero con un algoritmo **FFT** se puede obtener el mismo resultado con sólo $N \log N$.

La idea que permite esta optimización, es la descomposición de la transformada a tratar en otras más simples y éstas a su vez hasta llegar a transformadas de 2 elementos donde k puede tomar los valores 0 y 1. Una vez resueltas las transformadas más simples hay que agruparlas en otras de nivel superior que deben resolverse de nuevo y así sucesivamente hasta llegar al nivel más alto. Al final de este proceso, los resultados obtenidos deben reordenarse. [2]

En cuanto a nuestro código, la librería *numpy* proporciona el algoritmo necesario para realizar la fft de cualquier conjunto de datos, de una o dos dimensiones, por lo tanto, como se muestra en la Figura 6. Tener el espectro de Fourier de toda la base de datos, se puede resolver con una simple linea.

```
#fft for each row (axis=1)
Database_fft=abs(np.fft.rfft(Database,axis=1))
```

Figura 6. Fast Forier Transform of Database.

el algoritmo de **FFT** puede realizar su procedimiento en cualquiera de los ejes cuando se le ingresa un arreglo de 2 o mas dimensiones, por lo tanto, al especificar $axis = 1$, le estamos diciendo al algoritmo que debe realizar la **FFT** por filas. Ademas, si utilizamos **rFFT**, estamos especificando que solo queremos obtener la parte real de la transformada.

V-A3. Archivos .csv: Los archivos CSV *comma-separated values* son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas y las filas por saltos de línea.[3]

Utilizamos este tipo de archivos en nuestro código para almacenar la base de datos en el disco duro y no tener que armarla cada vez que vamos a ejecutar algún comando que la necesite. Para ello utilizamos la librería *pandas* en la cual, mediante un dataframe [4] exportamos a un archivo .csv como se puede observar en la Figura 7.

```
In [8]: pd.DataFrame(Database_fft).to_csv('fft_Database.csv',index=None)

In [9]: pd.read_csv('fft_Database.csv')
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	220491	220492
0	40.374695	27.752444	31.811325	14.355251	16.732195	31.968340	29.428578	20.900938	12.563004	13.663472	...	0.142252	0.347153
1	14.383972	16.492736	53.467186	29.856403	71.223745	17.949417	37.167709	16.227122	43.212111	25.190619	...	0.211261	0.481067
2	4.876556	56.472384	55.144929	68.214935	32.857278	28.036319	43.649947	54.357485	59.083102	58.751716	...	0.040338	0.008621
3	58.269989	60.261792	259.105133	49.168906	324.628725	52.323873	331.167529	57.497620	54.085055	88.729614	...	0.576168	0.983638
4	46.300568	20.660282	41.891719	30.329573	25.328290	26.498722	22.349660	16.547474	27.690085	10.471977	...	0.037534	0.086674
5	0.643738	45.089008	40.693905	77.962783	34.117270	73.968804	50.138399	39.110131	53.017297	6.344041	...	0.058150	0.189773
6	583.621735	60.081253	293.810945	17.037987	554.446531	61.443962	280.045236	22.788227	356.018251	59.076265	...	0.253662	0.686839
7	849.206573	61.746974	78.004019	18.063171	411.144909	59.812354	367.102752	23.340185	95.385360	56.397789	...	0.365695	0.590531
8	52.847870	84.795636	91.808109	43.638181	116.094292	58.279844	125.519784	59.974230	115.135708	60.236518	...	0.058459	0.061735
9	50.115753	102.466708	43.981173	34.563745	58.773559	60.503140	118.774860	18.047319	23.815744	64.845921	...	0.234802	0.398541

Figura 7. Base de datos en formato .csv

[2]Wikipedia: la enciclopedia libre - Fast Fourier Transform

[3]Wikipedia: la enciclopedia libre - valores separados por comas

[4]marco de datos

V-A4. Vector de etiquetas: este es uno de los apartados mas importantes para el algoritmo de clasificación, ya que es la única forma de decirle al mismo cuantas clases existen y la clase a la que pertenece cada dato. en la Figura 8. se presenta la creación del vector de etiquetas.

```
gen=np.zeros(np.size(Database_fft,axis=0))
for i in range(len(genre)):
    if i==1: gen[40:80]=1
    if i==2: gen[80:120]=2
    if i==1: gen[120:160]=3
gen
```

Figura 8. Vector de etiquetas

lo que nos dice este segmento de código, es que los primeros 40 datos, pertenecen a la clase 0, es decir el genero Electrónica, los datos $41-80$ pertenecen a la clase 1, es decir el genero Rap y asi sucesivamente.

V-B. Algoritmo Random Forest

Inicialmente, tomamos toda la base de datos armada en el apartado anterior y la separamos en datos de entrenamiento y datos de prueba. luego, procedemos a aplicar el algoritmo.

Random Forest es un método versátil de aprendizaje automático capaz de realizar tanto tareas de regresión como de clasificación. También lleva a cabo métodos de reducción dimensional, trata valores perdidos, valores atípicos y otros pasos esenciales de exploración de datos. Es un tipo de método de aprendizaje por conjuntos, donde un grupo de modelos débiles se combinan para formar un modelo poderoso.

En Random Forest se ejecutan varios algoritmos de árbol de decisiones en lugar de uno solo. Para clasificar un nuevo objeto basado en atributos, cada árbol de decisión da una clasificación y finalmente la decisión con mayor "votos" es la predicción del algoritmo.

Unos de los beneficios que más llama la atención es el poder de manejar grandes cantidades de datos con mayor dimensionalidad. Puede manejar miles de variables de entrada e identificar las variables más significativas, por lo que se considera uno de los métodos de reducción de dimensionalidad. Además el modelo muestra la importancia de la variable, que puede ser una característica muy útil.

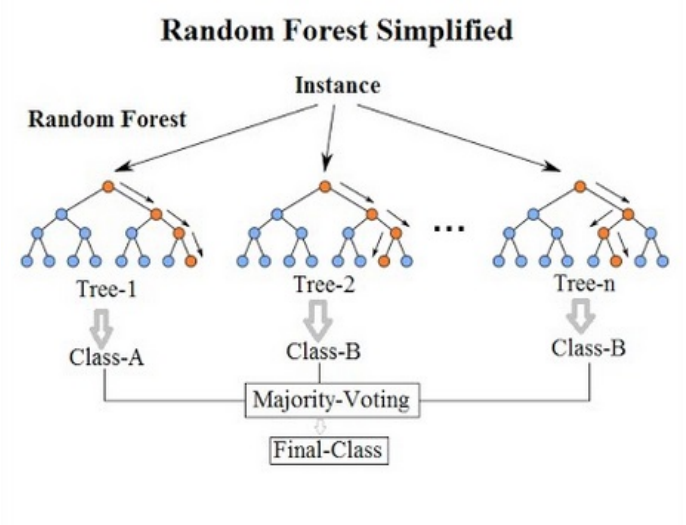

Figura 9. Random Forest Classifier.

En nuestro código implementamos el método de clasificación *Random Forest* el numero de veces que sea necesario, para obtener una precisión mayor al $70\,\%$. Esto se puede apreciar en la Figura 10.

```
accuracy = np.array([])
while True:
    rfc = RandomForestClassifier()
    rfc.fit(X_train,y_train) # fit and Standardize data
    np.append(accuracy,rfc.score(X_test,y_test))
    if rfc.score(X_test,y_test) > 0.7:
        break

print('Maximum accuracy for training set is :',rfc.score(X_test,y_test)*100)
```

Figura 10. Random Forest Classifier código.

Donde X_{train} se refiere a los datos de entrenamiento de la base de datos y Y_{train} se refiere a las etiquetas de dichos datos. X_{test} se refiere a los datos de prueba y Y_{test} a las etiquetas de los mismos.

Una vez finalizado el entrenamiento del modelo procedemos probarlo utilizando los datos de prueba, para ello, aplicamos una matriz de confusión entre las etiquetas reales y las clases hechas por el modelo, los resultados obtenidos se muestran en la Figura 11.

Como se observa en la Figura 11. Existían 8 datos de prueba para cada genero musical. La clasificación mas fuerte de estos datos se presento en el genero *Electronica* debido a que se clasificaron 7 datos y el modelo solo se

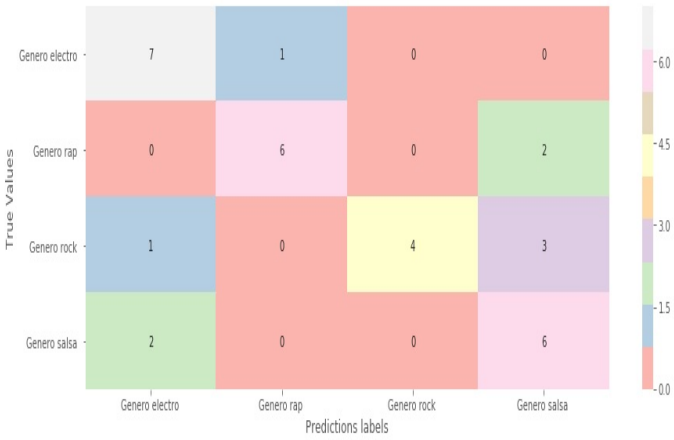

Figura 11. Matriz de confusión de los datos de prueba.

equivoco en uno. La clasificación mas débil del modelo, se presento en el genero de *Rock*, debido a que solo se clasificaron correctamente 4 datos, lo que representa la mitad de el total.

Realizamos la misma matriz de confusión entre el vector de etiquetas y las predicciones realizadas por el modelo a toda la matriz base de datos (160 canciones). Los resultados obtenidos se muestran en la Figura 12.

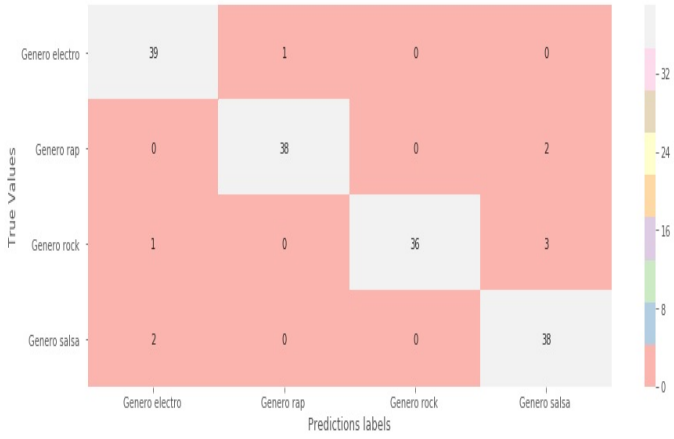

Figura 12. Matriz de confusión de la base de datos.

De 40 canciones existentes por cada genero, el modelo entrenado, clasifico correctamente, en el mejor de los casos 39 canciones (*Electrónica*). Nuevamente, la clase mas débil en cuanto a predicción fue el genero *Rock*, ya que se clasificaron 36 canciones y se presento el mismo error de 4 canciones que en la Figura 11. Estos errores de clasificación del modelo, corresponden a las canciones que presentan mayor varianza con respecto a la media de cada genero. Cabe aclarar que ningún sistema de clasificación es perfecto y en ocasiones se presentan este tipo de errores debido a la aleatoriedad de los datos. El reporte completo del modelo, se muestra en la Figura 13.

	precision	recall	f1-score	support
0.0	0.93	0.97	0.95	40
1.0	0.97	0.95	0.96	40
2.0	1.00	0.90	0.95	40
3.0	0.88	0.95	0.92	40
micro avg	0.94	0.94	0.94	160
macro avg	0.95	0.94	0.94	160
weighted avg	0.95	0.94	0.94	160

Figura 13. Reporte generado por el modelo.

Finalmente, se procede a guardar el modelo en un archivo *.pkl* para la etapa de validacion del sistema.

VI. Validación del sistema

Para el desarrollo de este apartado, utilizaremos una canción como ejemplo en cada una de las etapas, esta es *La rebelión - Joe Arrollo*, perteneciente al genero *Salsa*. Esta canción, no se encuentra dentro de la base de datos, por lo tanto, la utilizaremos para realizar una predicción con un ejemplo real. Como mencionamos en el apartado VI. la cancion debe estar en formato *.wav* para que pueda ser procesada en python, por lo tanto, este es el primer paso a a seguir, ademas, por simplicidad, se realiza el recorte a 5 segundos antes de que el archivo sea guardado.

VI-A. Modulación

Se realiza la implementación de un sistema de modulación digital utilizando la técnica *Double Sideband with Suppried Carrier*. en la Figura 14. Se muestra el diagrama de bloques del sistema de modulación.

Implementando esta técnica de modulación mediante código de python, graficamos el espectro de la señal modulada que se desea transmitir. el espectro de $X_{dsb-sc}(\omega)$ se muestra en la Figura 15.

Una vez realizada la modulación de la señal de entrada, esta se transmite de forma digital y luego recibe en una etapa de demodulación.

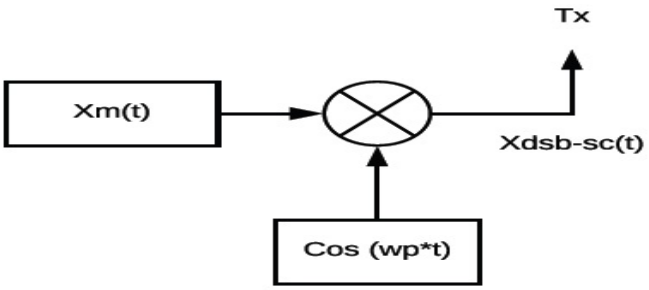

Figura 14. Double Sideband with Suppried Carried Modulation.

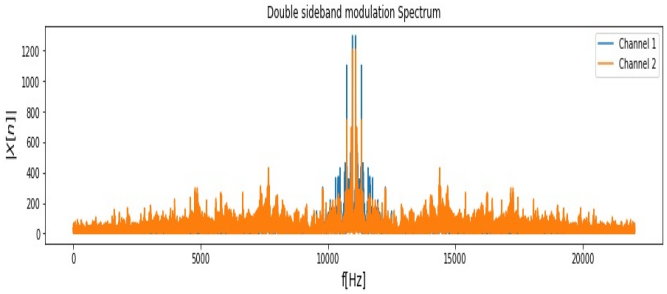

Figura 15. Double Sideband with Suppried Carried Spectrum.

VI-B. *Demodulación*

Para realizar la demodulación de la señal recibida utilizamos la técnica de *Sinchronus Detection*, la cual consiste en recibir la señal, multiplicarla nuevamente por la señal portadora y posteriormente, pasarla por una etapa de filtrado (*Low Pass*) para recuperar la señal original, este proceso, se muestra en la Figura 16.

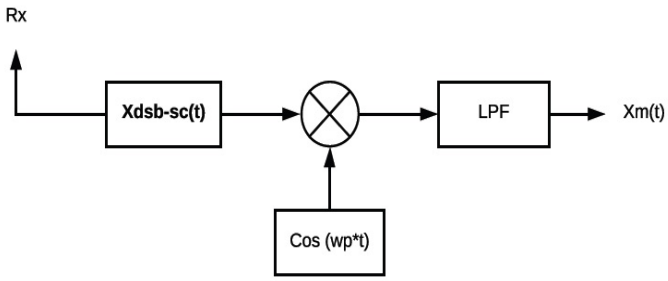

Figura 16. Sinchronus Detection Demodulation.

Implementando esta técnica en la etapa de demodulacion, obtenemos como resultado el espectro de la señal mensaje, lo cual se muestra en la Figura 17.

Una vez realizadas las etapas de modulación y demodulacion digital de la señal de entrada, procedemos a guardar la señal obtenida en un archivo con extensión *.wav*. Para ello se utiliza la librería soundfile que contiene la función .write para escribir archivos de audio.

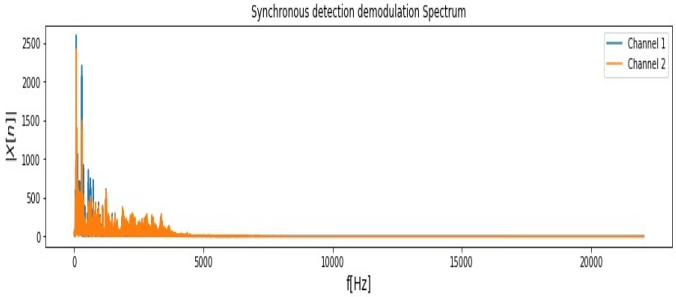

Figura 17. Sinchronus Detection Demodulation Spectrum.

VI-C. *Clasificación*

Por ultimo en esta etapa, procedemos a cargar el modelo **Random Forest.pkl** generado en el apartado V-B, para realizar la predicción de la señal obtenida. Se aplica el mismo procedimiento aplicado en la etapa V-A para el tratamiento de una nueva canción: Leer la canción con extensión *.wav* y concatenar los dos canales en un único arreglo unidimensional. luego se procede a realizar la predicción de la señal y a reproducir una canción de la base de datos a la cual pertenece la misma utilizando la librería *playsound*. Todo este procedimiento se puede observar en la Figura 18.

Predict with the random forest model

```
In [9]: rfc = joblib.load('RandomForest.pkl')              # Load the Random Forest model
        x,a = sf.read(dem)                                # Read the song after transmision
        xsong=np.concatenate((x[:,0],x[:,1]),axis=None)   # Concatenate left and right channel
        Xsongw=abs(np.fft.rfft(xsong))                    # Spectrum of concatenate song
        xpred=np.zeros((2,len(Xsongw)))                   # Song matrix with shape (2,data) for the prediction
        xpred[0,:]=Xsongw
        xpred[1,:]=Xsongw
        prediction=genre.get(int(rfc.predict(xpred)[0]))  # Predict genre of the song
        z  = random.randrange(40)                         # Select a random song in the database clasification
        print('The genre of the song is: %s \nPlay song of Database: %s'
              %(prediction,'Track_'+str(z)+'.wav'))
        playsound('Database/'+prediction+'/wavfiles/Track_' + str(z) + '.wav') # Play song

        The genre of the song is: Salsa
        Play song of Database: Track_25.wav
```

Figura 18. Predicción del modelo.

Como se puede observar en la Figura 18, el modelo predijo correctamente una nueva señal de entrada.

VII. Conclusiones

- Actualmente, existen muchos sistemas de reconocimiento y clasificación de canciones que se basan en algoritmos parecidos al desarrollado en este informe, sin embargo, estos algoritmos son mucho mas poderosos ya que presentan robustez frente al

ruido, algoritmos de aproximación para disminuir el error obtenido por el modelo, y además de reconocer géneros musicales, también pueden determinar el nombre de la canción y toda la información pertinente a la misma.

- Es indispensable, estandarizar los parámetros de entrada de las canciones para armar la base de datos, esto con el fin de facilidad de entendimiento y procesamiento, generalmente, los parámetros a estandarizar para todos los datos, son frecuencia de muestreo y tiempo inicial/final de la muestra tomada.
- Es posible utilizar la representación espectral de una canción para clasificar esta misma por géneros musicales, sin embargo, si se quiere mayor precisión en las predicciones, es necesario tener en cuenta otros parámetros como acusticidad, BPM, volumen, etc.
- La máxima precisión obtenida por nuestro modelo fue de $95\,\%$, un valor aceptable para la cantidad de datos que teníamos.