RECONOCIMIENTO DE VOZ PARA IDENTIFICAR EL RANGO DE EDAD DEL HABLANTE

Por: Enzo Camargo, Max Carrasco, Miguel Ruiz, Jhon Vargas

Facultad de Ciencias, Universidad Nacional de Ingeniería, Lima, Perú

Lima, 27 de marzo de 2021

RESUMEN

Uno de los problemas en los sistemas de reconocimiento automático de hablante son los cambios en la voz debido a factores ambientales como ruidos externos, por esfuerzo vocal al alzar o disminuir la voz intencionalmente o ronquera o al usar acentos del habla inusuales que corresponden a regiones específicas o también a la edad de las personas. Este último punto es analizado en este trabajo con el que se pretende encontrar aquellas características que permitan identificar la edad de las personas a partir de su voz identificada.

Palabras clave: Reconocimiento de voz, Clasificación de audios, red neuronal artificial.

I. INTRODUCCIÓN

Dentro del reconocimiento de voz otra de las partes más importantes es el análisis del audio mediante la extracción de sus características qué es un proceso esencial para el reconocimiento de patrones y las tareas de aprendizaje automático. Estas características tienen que ser informativas con respecto a las propiedades deseadas de la data original. Estas características pueden también ser vistas como un procedimiento de reducción de ratios porque deseamos que los algoritmos sean usados en un número menor de características. En nuestro caso se han usado 226 archivos de audio de tres rangos de edad a las cuales se les han extraído sus características para entrenar un sistema que luego permita, mediante el uso de otros archivos, identificar su grupo de edad indicando su precisión.

II. EL PROBLEMA

Los centros de llamadas del sistema financiero necesitan identificar a las personas con las cuales hablan, uno de estos variables son la edad del cliente identificado la que tiene que coincidir con la edad registrada en sus bases de datos. En ayuda de este procedimiento se desea saber con cierta precisión el rango de edad a la que pertenece el cliente que se está comunicando ya que de esta manera esta variable permitiría aumentar la precisión de la identidad del hablante y dificultar posibles fraudes por suplantación sobre todo cuando se realiza comunicaciones de voz de manera no presencial .

Según la CPAL (Centro Peruano de Audición y Lenguaje) frecuentemente si es posible acertar la edad de las personas por su voz, con un error promedio de 5 años. Las características de los órganos que producen la voz y el habla cambian a lo largo de la vida; los niños tienen voces finas y parecidas en cuanto al sexo; en la adolescencia la voz de los niños se torna más grave (gruesa), mientras que en las niñas cambian menos. Hombres y mujeres adultos tienen voces bastantes diferentes. Después de los 60 años, muchas voces masculinas quedan un poco más agudas (finas) y las femeninas tienden a quedar más graves (gruesa).

III. DISEÑO DE LA ARQUITECTURA

Para poder capturar las señales de audio, crear el dataset, modelar, entrenar y evaluar se tuvo que hacer el diseño de la arquitectura del proyecto para poder cumplir estos objetivos. En la Tab 1. se indican los componentes:

Item	Características
Hardware	Smartphone: Xiomi Redmi Pro Laptop: Asus
Software	OS: Linux/Debian 64bit Language: python Tool: ffmepg
Comunicaciones	Sensor: Smartphone

Algoritmo ANN	MLPClassifier

Tab 1.

En la Fig 1., podemos ver las unidades de la arquitectura:

1. Unidad de Adquisición: Los componentes principales son el smartphone y laptop. El objetivo principal es usar el smartphone cómo sensor de voz y la laptop cómo recolector de audio.
2. Unidad de Entrenamiento: Los componentes principales son la laptop y los algoritmos usados para crear el modelo, evaluar y predecir.

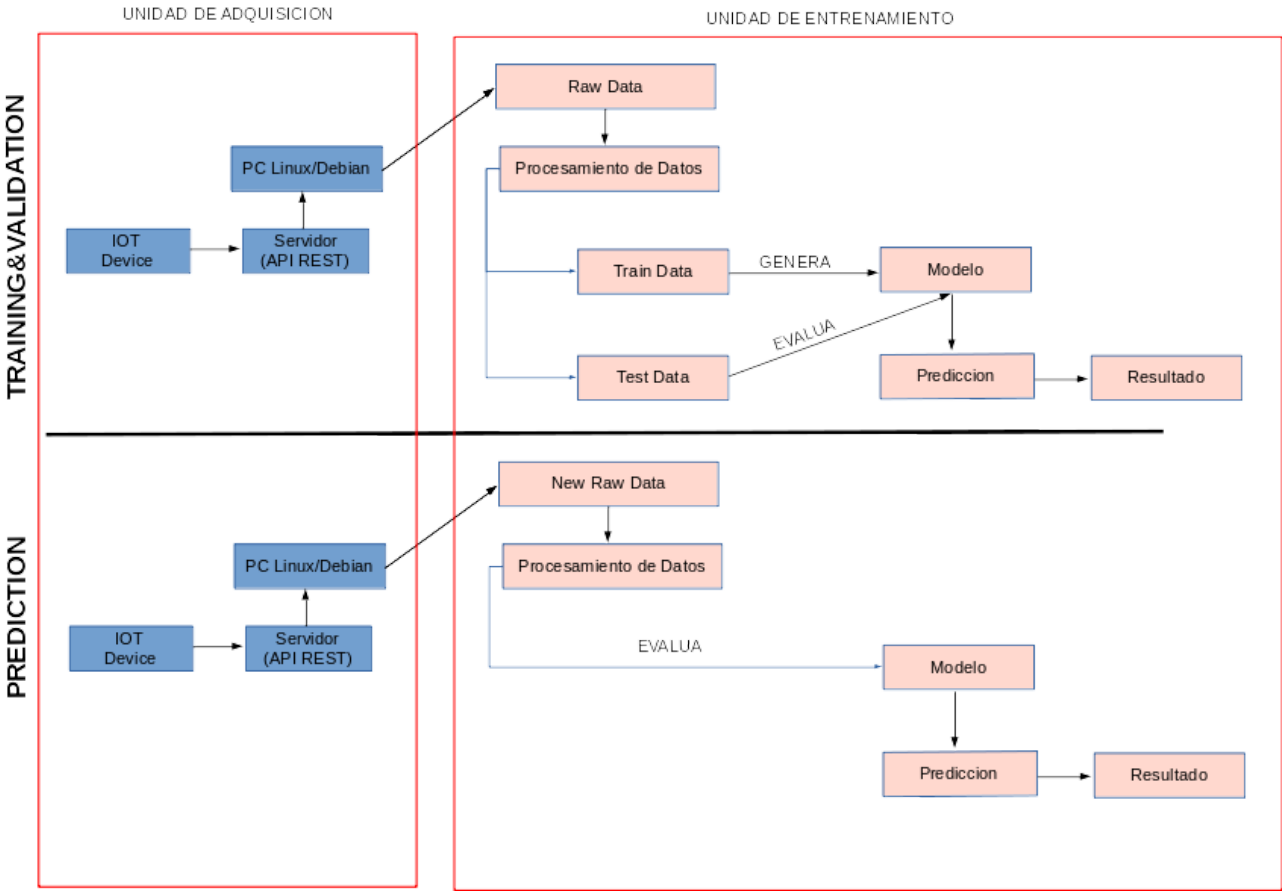


Fig 1.

IV. DISEÑO DEL EXPERIMENTO

Para poder cumplir con el objetivo, debemos realizar 2 tipos de experimentos para poder crear el modelo y validar el modelo respectivamente.

1. Experimento 1 (TRAINING & VALIDATION):
    a. Requerimientos:
        i. El experimento consiste en reconocer las siguientes categorías:
            - Rango 1: 10 a 14 años
            - Rango 2: 15 a 20 años
            - Rango 3: 30 a 60
        ii. Para el reconocimiento se usará la unidad de adquisición de la Fig.1.
        iii. Se usará un muestreo a 48kHz
        iv. Smartphone con sensor de audio.
    b. Experimento:
        i. Se procede a realizar el muestreo de la señal mediante el smartphone.
        ii. Luego se procede a una segmentación de audio recolectado para poder leer las frecuencias, esto se realiza localmente en PC/Laptop.
        iii. Preparación de datos para módulos python librosa, audio segment, pandas, numpy,
        iv. Modelado y evaluación con módulo python sklearn, pickle.
2. Experimento 2 (PREDICTION):
    a. Requerimientos:
        i. El experimento consiste en reconocer los rangos de edad usado el modelo ya creado en el experimento

1.
ii. Para el reconocimiento se usará la unidad de adquisición y la unidad de predicción de la Fig.1.
iii. Se usará un muestreo a 48kHz
iv. Smartphone con sensor de audio.

b. Experimento:
i. Se procede a realizar el muestreo de la señal mediante el smartphone.
ii. Luego se procede a una segmentación de audio recolectado localmente en PC/Laptop.
iii. Preparación de datos para módulos python librosa, audio segment, pandas, numpy,
iv. Predicción usando los módulo python sklearn, pickle.

En la Fig 1., podemos ver los experimentos que debemos realizar. La primera parte está relacionada con la creación del modelo. La segunda parte está relacionada con poner a prueba el modelo, predicción.

V. DAQ

El sistema de adquisición de datos para crear el experimento consiste hacer las grabaciones de audio usando de sensor al smartphone los cuales serán enviado a un servidor API y mediante una PC/Laptop podremos descargar estos archivos de audio(raw audio) localmente para su posterior procesamiento. En la Fig 2., podemos ver el flujo de adquisición de datos.

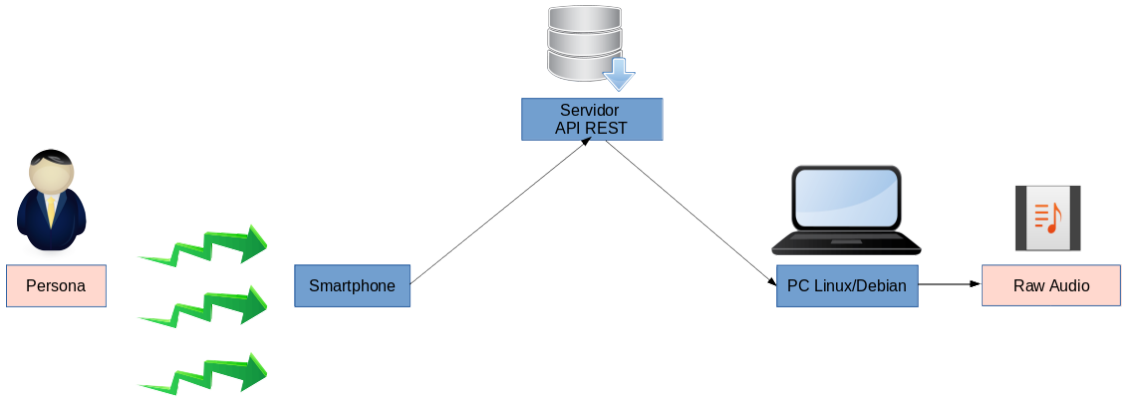


Fig 2.

VI. METODOLOGÍA DEL TRABAJO

Para este trabajo se ha considerado tomar muestras de archivos de audio de personas de sexo masculino divididas en tres rangos de edad:

- Rango1: 10 a 14 años
- Rango2: 15 a 20 años
- Rango3: 30 a 60

Los archivos de audio deben de tener una longitud fija y estar en formato wav. El nombre de cada archivo indicará el rango al que pertenecen. Estos archivos se colocarán en una carpeta de google drive y mediante el uso del lenguaje python, usando google colab, se realizarán los procedimientos de etiquetado, extracción de sus características, entrenamiento del modelo. Luego de obtener cierta precisión mediante el análisis de diferentes resultados se procede a probar la capacidad de predicción del modelo mediante el uso de otros archivos de audio pertenecientes a los rangos indicados obteniéndose al final una medida de su precisión.

En la Fig 3, detallamos la metodología de trabajo:

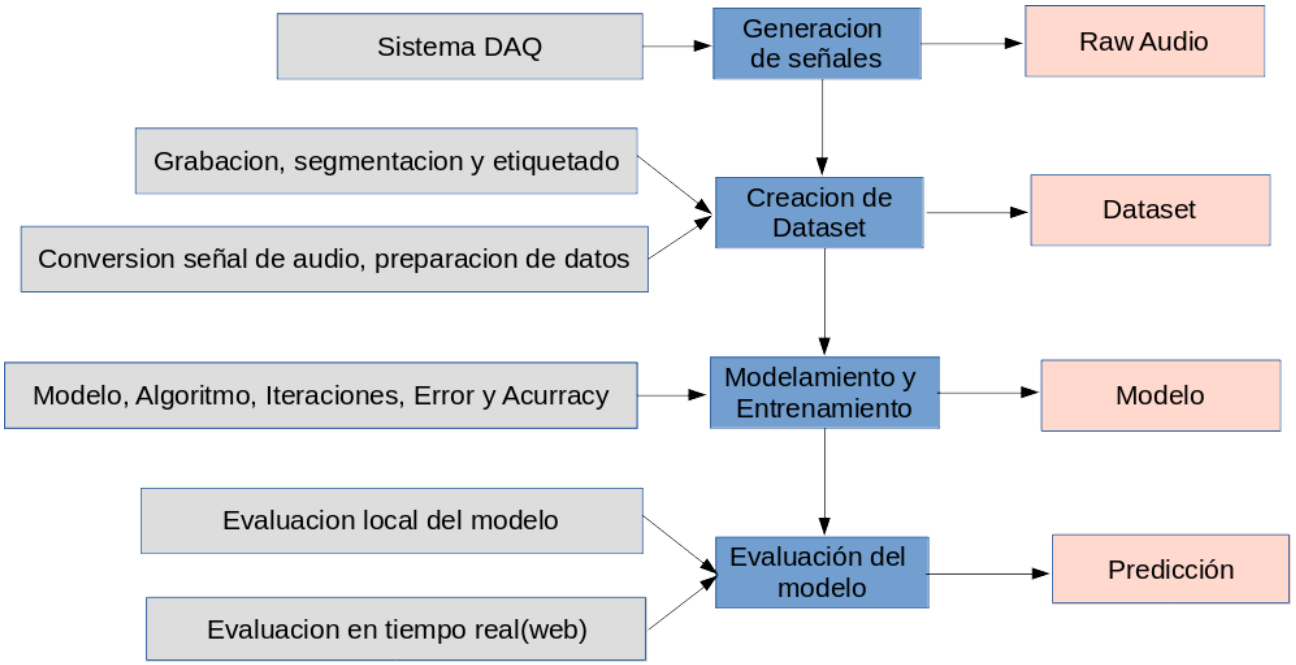


Fig 3.

VII. DESARROLLO DE LA SOLUCIÓN

Se usan 226 archivos de audio wav cada uno de 1 segundo de longitud. Contiene las voces masculinas de tres rangos de edad:

- Rango1: 10 a 14 años
- Rango2: 15 a 20 años
- Rango3: 30 a 60

Estos archivos se pueden consultar en el siguiente link:

https://drive.google.com/drive/folders/1X4TramhO24KE2b-F86zH0nIKJXdw5YHR?usp=sharing

Al código python, en google colab, se puede acceder en el siguiente link:

https://colab.research.google.com/drive/186o4HfXkbirElIJ1zAQHVNcwDkXB-KaT?usp=sharing

Para el desarrollo de la solución hemos diseñado las siguientes etapas:

1. CREACIÓN DEL DATASET

El desarrollo de esta etapa tiene el siguiente flujo. En la Fig 4., podemos ver las siguientes partes:

- Entrada: El audio está recolectado del smartphone que es el sensor de voz y se recolecta en la PC.
- Salida: 1ra salida sería el espectrograma del audio y cómo 2da salida tenemos el dataset generado por las diversas herramientas de python.
- Herramientas: Se usó una Laptop ASUS 8G RAM, OS Linux/Debian. Se usaron cómo herramientas al reproductor VLC, ffmpeg del sistema operativo, python con modulos librosa, audiosegment, sklearn, entre otros.

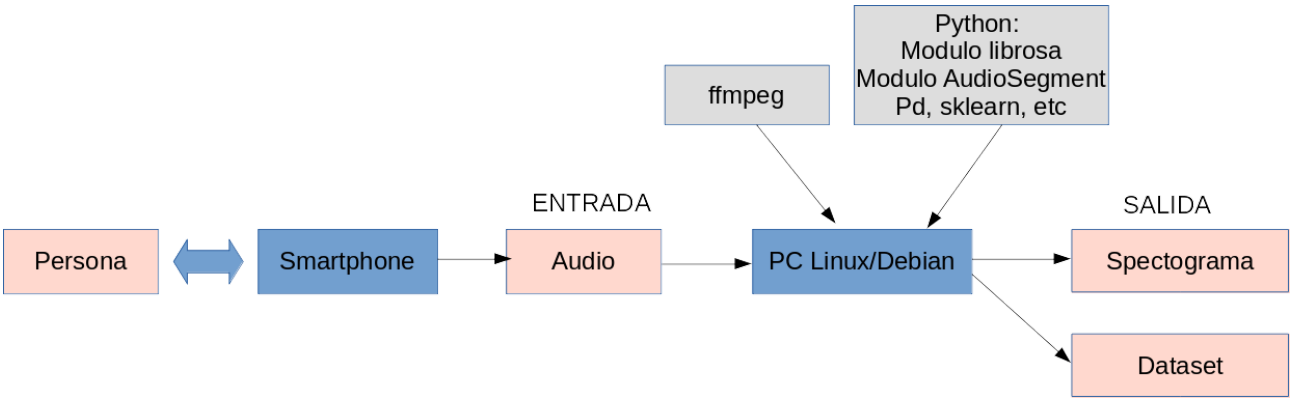


Fig 4.

Cómo primer paso se definen los rangos de voz a analizar. En la siguiente tabla Tab 2. Mostramos los 3 rangos:

Audio ID	Rango de Edad	ETIQUETA
audio1	10 años a 14 años	1
audio2	15 años a 20 años	2
audio3	30 años a 60 años	3

Tab 2..

Para la segmentación, se decidió usar particiones(ventanas) de 1seg tal que puedan cubrir todos los tipos de frecuencias y que también se genere una cantidad adecuada de datos en cada grupo. Luego se usó comandos ffmpeg para poder particionar los audios en 1seg:

- ffmpeg -i audio002_000.mp3 -f segment -segment_time 1 -c copy audio002_000_%03d_otros.mp3
- ffmpeg -i audio002_001.mp3 -f segment -segment_time 1 -c copy audio002_001_%03d_otros.mp3
- ffmpeg -i audio002_002.mp3 -f segment -segment_time 1 -c copy audio002_002_%03d_otros.mp3

Luego usaremos los audio en wav que es el formato que utiliza el módulo librosa en python que lo utilizamos para poder generar espectrogramas a partir de las señales de audio. En la Fig 5., podemos ver el código utilizado:

```
############################################################
################### Preparacion de Datos  ##################

# Crear expectogramas a partir de los audios.
# 1. Convertir los audios.wav en png para poder crear los atributos.

# Definimos el color map que usaremos para crear el espectograma
cmap = plt.get_cmap('inferno')
plt.figure(figsize=(8,8))

# ./TestData es la ruta donde esta todos los audios etiquetados Audio00x_xyz_sloganw.
path = r'./RawData'
extension = '.wav'


for root, dirs_list, files_list in os.walk(path):
    for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
            file_name_path = os.path.join(root, file_name)
            name=os.path.splitext(file_name)[0]
            y, sr = librosa.load(file_name_path, mono=True, duration=5)
            plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='default', mode='default', sc
            plt.axis('off');
            plt.savefig(f'{file_name_path[:-3]}png')
            plt.clf()
############################################################
```

Fig 5.

Los espectrogramas nos servirán para poder extraer las siguientes características de la señal y de este modelo la podemos usar para crear los atributos del dataset. En la Fig 6., podemos ver un ejemplo de un espectrograma de una señal de audio.

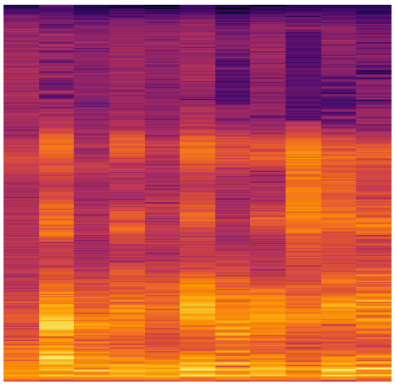


Fig 6.

Para crear los atributos, usamos el módulo librosa en python mediante el cual pudimos crear los siguientes atributos:

- Coeficientes de frecuencia Cepstral MEL (MFCC):
- Centroide espectral: Medida simple de posición y forma espectral. El centroide espectral es el centro de "gravedad" del espectro.
- Tasa de cruce por cero:
- Frecuencias cromáticas:
- Caída Espectral (Espectral rolloff): Representa la frecuencia por debajo de la cual un cierto porcentaje (generalmente 80% -90%) de la distribución de magnitud del espectro se concentra en el espectro.
- Chroma: Categoriza los tonos de manera significativa (a menudo en doce categorías). Captura las características armónicas y melódicas de la música, de manera invariante a cambios en el timbre y el ruido.
- Error Cuadrático Medio (RMSE) : Medida de medición de potencia de audio eléctrica transferida de un amplificador de audio a un altavoz, que se calcula en watts.
- Ancho de Banda Espectral (spectral_bandwidth): Es el ancho de la línea horizontal en el espectrograma de una onda sinusoidal pura.

En la Fig 7., podemos ver el código de esta creación de atributos usando el módulo librosa:

```
# Extraemos las caracteristicas desde el Espectograma que se convertiran en atributos del dataset:
# 1. Mel-frequency cepstral coefficients (MFCC)
# 2. Spectral Centroid
# 3. Zero Crossing Rate
# 4. Chroma Frequencies
# 5. Spectral Roll-off.
path = r'./RawData'
extension = '.wav'
for root, dirs_list, files_list in os.walk(path):
    for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
            file_name_path = os.path.join(root, file_name)
            audio = file_name_path
            y, sr = librosa.load(audio, mono=True, duration=30)
            rmse = librosa.feature.rms(y=y)
            chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
            spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
            spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
            rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
            zcr = librosa.feature.zero_crossing_rate(y)
            mfcc = librosa.feature.mfcc(y=y, sr=sr)
            p_slogan = identificar_parametros(file_name)
            to_append = f'{file_name_path} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np.mean
            for e in mfcc:
                to_append += f' {np.mean(e)}'
            to_append += f' {p_slogan}'
            file = open('dataset.csv', 'a', newline='')
            with file:
                writer = csv.writer(file)
                writer.writerow(to_append.split())

# Cargamos y leemos los datos en la variable pandas data.
# Contiene las etiquetas y atributos.
data = pd.read_csv('dataset.csv')
data.head(10)
```

Fig 7.

Y con esto generamos el primer dataset (draft) que afinaremos conforme avancemos con los pasos de preparación y entendimiento de datos. En la Fig 8., vemos el comando “data.head(10)”, que nos muestra las diez primeras columnas del dataset.

	filename	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rate	mfcc1	mfc
0	./RawData/audio003_rango3_1_026.wav	0.320003	0.102224	797.608997	902.763182	1342.399458	0.042980	-297.560272	209.7571
1	./RawData/audio003_rango3_3_036.wav	0.394428	0.142272	1767.396261	1585.304391	3411.299827	0.114125	-155.402115	145.7466
2	./RawData/audio003_rango3_2_028.wav	0.371871	0.058041	1471.973429	1917.454245	2926.558061	0.066606	-292.720398	138.8751
3	./RawData/audio003_rango3_2_031.wav	0.341493	0.050442	1315.071153	1816.271286	2605.267192	0.051133	-334.287292	146.0216
4	./RawData/audio003_rango3_2_041.wav	0.393041	0.054884	1717.254696	1980.082841	3374.840199	0.081865	-314.865295	127.9423
5	./RawData/audio003_rango3_3_052.wav	0.366415	0.180120	1306.104372	1402.592194	2722.726718	0.061268	-153.250061	163.9823
6	./RawData/audio002_rango2_1_010.wav	0.273155	0.031712	1927.308901	1924.916365	3397.352184	0.111828	-306.823792	127.9699
7	./RawData/audio003_rango3_2_024.wav	0.301227	0.005747	2276.702347	2520.614243	5069.845858	0.102927	-412.320709	96.0391
8	./RawData/audio003_rango3_1_009.wav	0.387961	0.076727	1516.805170	1577.250461	3090.504039	0.065940	-246.963852	148.7175
9	./RawData/audio003_rango3_3_054.wav	0.387725	0.140499	1216.732695	1364.382438	2412.452836	0.056041	-171.807510	181.6993

10 rows × 28 columns

Fig 8.

Para la preparación y entendimiento de los datos, ejecutaremos los siguientes pasos:

1. Eliminación de NULLs or NA

Primero validamos si existen datos NULL o NA. En la Fig 9., podemos ver el código usado. Para este fin usaremos los comandos “data.isnull().valuesany()” y “data.isna().sum().sum()”. Segun vemos no existen datos NULL o NA.

```
###### 1. NULLs or NA #######
# Verificamos si existen datos Null, NA en el dataset.
data = pd.read_csv('dataset.csv')
print("Datos Null y NA en el dataset")
print("============================== \n")
print(data.isnull().sum())
print("\n")
print("Datos Null en todo el dataset: ", data.isnull().values.any())
print("Datos NA en todo el dataset: ", data.isna().sum().sum())
```

```
Datos Null y NA en el dataset
==============================

filename              0
chroma_stft           0
rmse                  0
spectral_centroid     0
spectral_bandwidth    0
rolloff               0
zero_crossing_rate    0
mfcc1                 0
mfcc2                 0
mfcc3                 0
mfcc4                 0
mfcc5                 0
mfcc6                 0
mfcc7                 0
mfcc8                 0
mfcc9                 0
mfcc10                0
mfcc11                0
mfcc12                0
mfcc13                0
mfcc14                0
mfcc15                0
mfcc16                0
mfcc17                0
mfcc18                0
mfcc19                0
mfcc20                0
clase                 0
```

Fig 9.

2. Eliminación de Outliers o Datos extremos

Para eliminar los datos extremos, analizaremos la asimetria de los datos. Para esto usaremos el código en python “data.skew()” donde podemos ver en la Fig 10. que algunos datos tienen algún aplanamiento(skew>0 y skew<0). Segun la grafica podemos ver que existen varios datos que tienen datos extremos cómo son “spectral_centroid, zero_crossing_rate, rolloff, mfcc7”.

```python
###### 2. Outlayers or Datos extremos #######
# Verificamos si existen datos extremos en el dataset.
data = pd.read_csv('dataset.csv')

print("Asimetria de los datos:\n", data.skew())

```

```
Asimetria de los datos:
 chroma_stft           0.381479
rmse                  0.747852
spectral_centroid     1.383628
spectral_bandwidth    0.853958
rolloff               1.315135
zero_crossing_rate    1.189115
mfcc1                 0.126746
mfcc2                -0.618317
mfcc3                -0.011009
mfcc4                 0.036676
mfcc5                -0.215819
mfcc6                -0.919414
mfcc7                -1.169556
mfcc8                -0.269864
mfcc9                -0.026281
mfcc10               -0.203744
mfcc11                0.496826
mfcc12                0.156416
mfcc13               -0.002128
mfcc14                0.345349
mfcc15               -0.306382
mfcc16               -0.815996
mfcc17                0.502985
mfcc18               -0.044285
mfcc19                0.392453
mfcc20                0.584918
dtype: float64
```

Fig 10.

Esta dispersión también la podemos seguir analizando con los histogramas de cada columna(atributo) tal cómo se muestra en la Fig 11. Y Fig 12., podemos validar esta información:

```python
#Filtramos las columanas para poder analizar por grupos
columns = data.columns
print(data.columns)
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
cols = [c for c in data.columns if data[c].dtype in numerics]

cols1 = ['chroma_stft', 'rmse', 'spectral_centroid', 'spectral_bandwidth', 'rolloff', 'zero_crossing_rate']
cols2 = ['mfcc1', 'mfcc2', 'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10']
cols3 = ['mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17', 'mfcc18', 'mfcc19', 'mfcc20']

#Generamos 3 tipos de dataset para entender la dispercion que tienen los datos
data1 = data[cols1]
data2 = data[cols2]
data3 = data[cols3]

ax1 = data1.hist(bins= 10, alpha=0.5)
ax1 = data2.hist(bins= 10, alpha=0.5)
ax1 = data3.hist(bins= 10, alpha=0.5)
plt.show()

```

Fig 11.

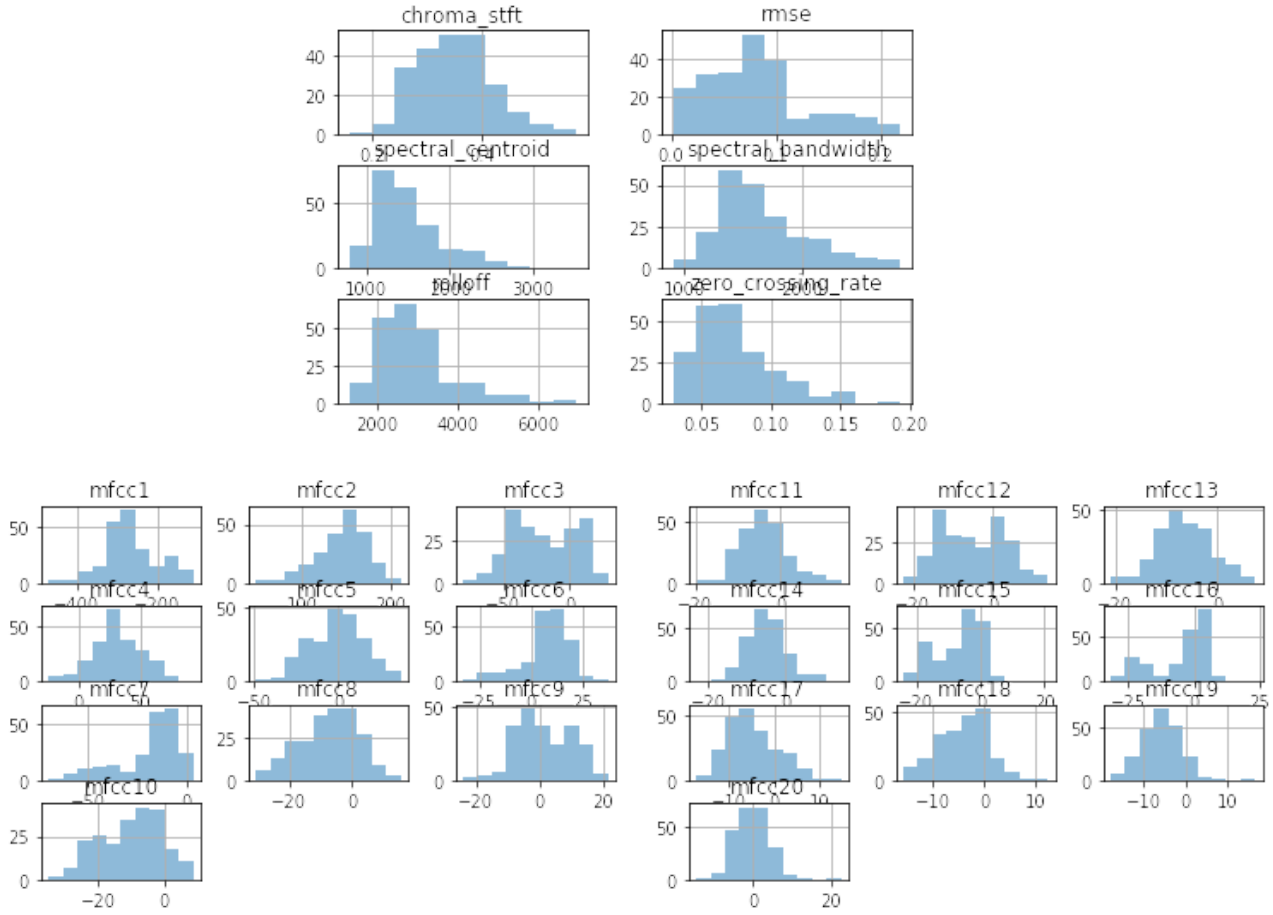


Fig 12.

Para poder eliminar los datos extremos usaremos los Quartiles con un IRQ = Q3-Q1. Dividimos la data en 4 cuartiles:

- Primer cuartil (Q1) = 25% de los datos.
- Segundo cuartil (Q2) = 50% de los datos
- Tercer cuartil (Q3) = 75 % de los datos.
- Cuarto cuartil(Q4) = 100% de los datos

Luego los valores extremos lo obtenemos con el siguiente filtro(intervalo), el cual usaremos para remover los outlier con python tal cómo se muestran en la Fig 13. En la Fig 14., vemos los outliers en boxplot antes de la remoción y después de la remoción sin los outliers.

Filtro: [Q1 - 1.5 * IQR, Q3 + 1.5 * IQR]

```
# Calculamos los quartiles y el IQR
Q0 = data1.quantile(0.00)
Q1 = data1.quantile(0.25)
Q2 = data1.quantile(0.50)
Q3 = data1.quantile(0.75)
Q4 = data1.quantile(1.00)

# Creamos el IRQ con el 3er y 1er quartil
IQR= Q3 - Q1

# Creamos el filtro que nos validara la existencia de valores extremos que este fuera del 1.5*IRQ en cada column
filtro = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)
outliers = data[filtro]

# Sacamos los datos extemos y actualizamos el dataset
data_new = data[~filtro]

# Actualizamos dataset
data = data_new

#Grafico de los datos extremos
plt.figure(figsize=(20,20))
outliers.boxplot()
plt.show()

#Grafica sin los datos extremos
plt.figure(figsize=(20,20))
data_new.boxplot()
plt.show()
```

Fig 13.

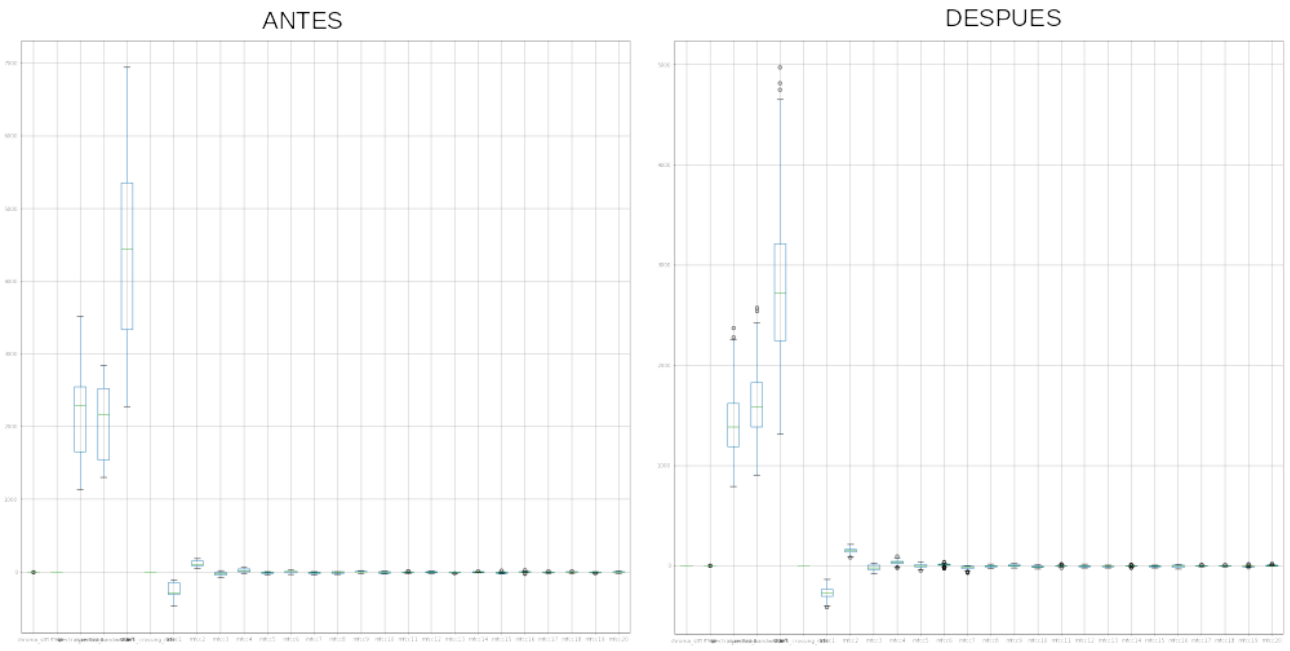


Fig 14..

Luego de esta etapa tenemos una nueva data que la usaremos para balancear los datos mediante las clases.

3. Balanceo de datos mediante las clases

Primero balancear los datos, primero verificaremos si las clases están balanceadas. Luego usaremos la técnica de Over-Sample que se encarga de rellenar en cantidades minoritarias. En la Fig15., podemos ver el código usado en python y tambien ver los cambios antes y después.

```python
###### 3. Balanceo de datos mediante las clases #######
# Primero verificamos que una gran diferencia entre la clase "otros" y las otras clases slogan1,2,3,4,5...

# Buscamos los valores de cada clase en base a filas
rango3, rango2, rango1 = data["clase"].value_counts()

# Segun esto podemos verificar que existe un gran desbalance entre la clase "otros" y las otras clases
# Cantidad de filas por tipo de clase
print("Cantidad de filas por Clase antes del balanceo: \n", data["clase"].value_counts())

#Creamos un filtro para cada clase
g1 = data.loc[data["clase"]=="rango1"]
g2 = data.loc[data["clase"]=="rango2"]
g3 = data.loc[data["clase"]=="rango3"]

# Luego aplicamos la tecnica de Over-sample, que es utilizada para rellenar a las cantidades minoritarias
# que en este caso son las clases slogan1, slogan2, ... para que se iguale en cantidad con la clase otros.
g1_over = g1.sample(rango3, replace=True)
g2_over = g2.sample(rango3, replace=True)
g3_over = g3.sample(rango3, replace=True)


# Balanceamos los datos
data_balanced = pd.concat([g1_over, g2_over, g3_over])
# Volvemos a verificar el desbalance entre la clase "otros" y las otras clases, y confirmamos que ya no existe
# el desbalance.
# Cantidad de filas por tipo de clase luego del balanceo
print("\n")
print("Cantidad de filas por Clase despues del balanceo: \n", data_balanced['clase'].value_counts())
print(data_balanced.shape)
data_balanced.head(10)

# Actualizamos dataset
data = data_balanced
```

```
Cantidad de filas por Clase antes del balanceo:
 rango3    179
rango2     18
rango1      3
Name: clase, dtype: int64


Cantidad de filas por Clase despues del balanceo:
 rango3    179
rango2    179
rango1    179
Name: clase, dtype: int64
(537, 28)
```

Fig 15.

4. Eliminación de columnas innecesarias

Luego hacemos un drop a las columnas innecesarias. En Fig 16., vemos el código usado.

```python
###### 4. Eliminacion de columnas innesesarias #######
# Eliminamos las columanas que no utilizamos como la columna "filename" que es usada como nombre del archivo.
data = pd.read_csv('dataset.csv')
data = data.drop(['filename'],axis=1)
data.head(10)
```

Fig 16.

5. Normalización y codificación de la clase

Normalizamos los datos y encodificamos la clase para poder hacer las predicciones. En la Fig 17., para esto usaremos el siguiente código en python.

```
###### 5. Normalizacion #######

X = data.iloc[:, :-1]
#print(np.array(data.iloc[:, :-1]))

minmax_scale = preprocessing.MinMaxScaler().fit(X)
X = minmax_scale.transform(np.array(data.iloc[:, :-1], dtype = float))

############ 6. Codificacion de la clase ##############

clase_list = data.iloc[:,-1]
encoder = LabelEncoder()
y = encoder.fit_transform(clase_list)
data["clase"] = y
print(data["clase"].value_counts())

data.head(10)
```

```
2    193
1     23
0     10
Name: clase, dtype: int64
```

	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rate	mfcc1	mfcc2	mfcc3	mfcc4	...	mfc
0	0.320003	0.102224	797.608997	902.763182	1342.399458	0.042980	-297.560272	209.757172	-0.672393	4.100792	...	-9.977
1	0.394428	0.142272	1767.396261	1585.304391	3411.299827	0.114125	-155.402115	145.746674	-52.355831	51.472836	...	-8.69
2	0.371871	0.058041	1471.973429	1917.454245	2926.558061	0.066606	-292.720398	138.875168	18.074665	27.647676	...	2.898
3	0.341493	0.050442	1315.071153	1816.271286	2605.267192	0.051133	-334.287292	146.021683	8.799952	21.486828	...	-0.316
4	0.393041	0.054884	1717.254696	1980.082841	3374.840199	0.081865	-314.865295	127.942375	8.356564	30.512653	...	-1.484
5	0.366415	0.180120	1306.104372	1402.592194	2722.726718	0.061268	-153.250061	163.982346	-43.389076	49.418858	...	-9.792
6	0.273155	0.031712	1927.308901	1924.916365	3397.352184	0.111828	-306.823792	127.969994	-22.938957	3.947174	...	4.021
7	0.301227	0.005747	2276.702347	2520.614243	5069.845858	0.102927	-412.320709	96.039131	-1.826294	15.271872	...	-0.484
8	0.387961	0.076727	1516.805170	1577.250461	3090.504039	0.065940	-246.963852	148.717545	-38.248676	54.997341	...	-5.439
9	0.387725	0.140499	1216.732695	1364.382438	2412.452836	0.056041	-171.807510	181.699356	-47.779079	51.377377	...	-16.738

10 rows × 27 columns

Fig 17.

2. MODELAMIENTO Y ENTRENAMIENTO

Para el modelamiento, cómo es un escenario de clasificación multistate, usaremos una red neuronal artificial de 3 capas, que utiliza una función de activación de salida softmax para poder manejar estos casos de clasificación multistate. Para trabajar la data de entrenamiento y test la dividiremos en 80%. 20% respectivamente. En la Fig 18., vemos la implementación del modelo:

```
# Creacion del modelo con el algoritmo MLPClassifier ANN con las siguientes caracteristicas
# 1. Uso funcion de activacion softmax en la ultima capa(Esta configuracion esta en la logica del algoritmo)
# 2. Uso maximo de 300 iteracion
# 3. Uso de 3 capas
clf = MLPClassifier(random_state=1, max_iter=300)

#######
# separa los datos para entrenamiento y para pruebas
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=1, test_size=0.2)
#print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# entrena clasificador
clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train,y_train)
print(clf)

# calcula probabilidades, solo el primero
y_test_prob = clf.predict_proba(X_test)
#print('X_test', X_test)
#print('y_test_prob', y_test_prob)

# predice resultados, solo el primero
y_test_pred = clf.predict(X_test)
#print('X_test', X_test)
#print('y_test_prd', y_test_pred)

# matriz de confusion
cmatrix = confusion_matrix(y_test, y_test_pred)
print(cmatrix)
plot_confusion_matrix(clf, X_test, y_test)
plt.show()

# Evaluacion del modelo
print("Score: ",  clf.score(X_test, y_test))
print("Acurracy: ", accuracy_score(y_test,y_test_pred)*100)
print("Recall: ", recall_score(y_test, y_test_pred, average=None))
print("Precision: ", precision_score(y_test, y_test_pred, average=None)*100)

print("Funcion de acctivacion en la ultima capa: ", clf.out_activation_)
print("Perdida obtenida durante el entranamiento")
print(clf.loss_)
print(clf.best_loss_)
print("Numero iteraciones")
print(clf.n_iter_)
print("Numero de capas")
print(dim(clf.coefs_))

#######
```

Fig 18.

En los resultados se muestran en la Fig 19. Y Fig 20.

- Score: 1.0
- Acurracy: 100.0
- Recall: [1. 1. 1.]
- Precision: [100. 100. 100.]
- Función de activación en la última capa: softmax
- Pérdida obtenida durante el entranamiento: 0.0059999091180605 63
- Número iteraciones: 201
- Número de capas: [2, 26, 100]

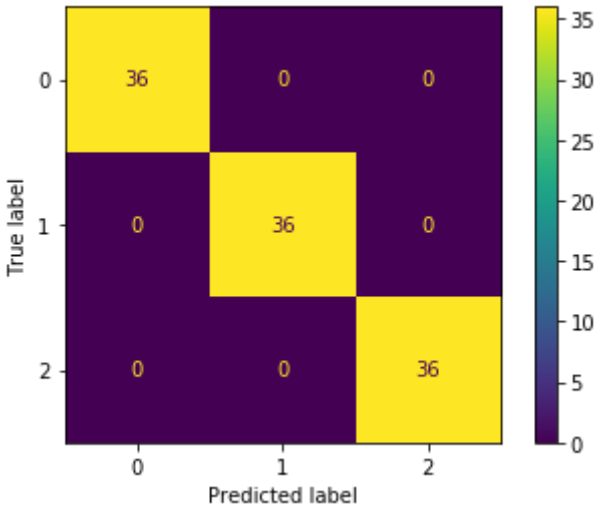


Fig 19..

Y la curva de error es la siguiente:

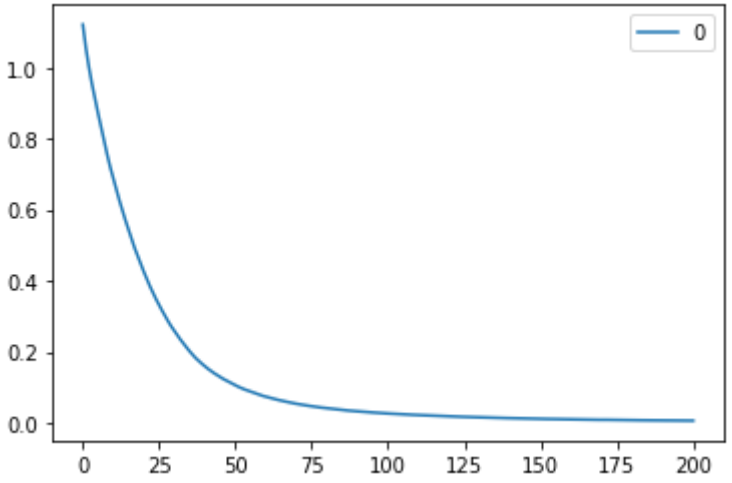


Fig 20.

3. EVALUACIÓN DEL MODELO

Para la evaluación del modelo usaremos el formato pickle el cual usaremos después para evaluar el modelo luego de su entrenamiento. Con el comando “pickle.dump” podemos crear el modelo y con el comando “pickle.load” cargamos el modelo.

```
1  ################################################################
2  ################### Evaluacion del Modelo ######################
3
4  # Creacion del clasificador
5  pickle.dump(clf, open('classifier.pkl', 'wb'), protocol=4)
6
7  # Cargamos el modelo para su evaluacion
8  clf = pickle.load(open('classifier.pkl', 'rb'))
9
```

Fig 21.

Después generamos el audio con el smartphone y se lo pasamos al modelo para que genere el dataset de este dato. En la Fig 22., podemos ver el código usado y el . La salida de este paso es generar un dataset(evaluación.csv) sin clase el cual se usará para evaluar el modelo.

```
path = r'./Test'
extension = '.wav'
for root, dirs_list, files_list in os.walk(path):
    for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
            file_name_path = os.path.join(root, file_name)
            name=os.path.splitext(file_name)[0]
            y, sr = librosa.load(file_name_path, mono=True, duration=5)
            plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='default', mode='default', sc
            plt.axis('off');
            plt.savefig(f'{file_name_path[:-3]}png')
            plt.clf()

header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth rolloff zero_crossing_rate'
for i in range(1, 21):
    header += f' mfcc{i}'
header = header.split()

# Extraer caracteristicas desde el Espectograma: Mel-frequency cepstral coefficients (MFCC), Spectral Centroid,
file = open('evaluacion.csv', 'w', newline='')
with file:
    writer = csv.writer(file)
    writer.writerow(header)

extension = '.wav'
for root, dirs_list, files_list in os.walk(path):
    for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
            file_name_path = os.path.join(root, file_name)
            audio = file_name_path
            y, sr = librosa.load(audio, mono=True, duration=30)
            rmse = librosa.feature.rms(y=y)
            chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
            spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
            spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
            rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
            zcr = librosa.feature.zero_crossing_rate(y)
            mfcc = librosa.feature.mfcc(y=y, sr=sr)
            to_append = f'{file_name_path} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np.mean
            for e in mfcc:
                to_append += f' {np.mean(e)}'
            file = open('evaluacion.csv', 'a', newline='')
            with file:
                writer = csv.writer(file)
                writer.writerow(to_append.split())
```

Fig 22.

En la Fig 23., podemos ver el dataset(evaluación.csv) creado sin la columna “clase”:

```
# Procesamiento de datos: Cargar la data del CSV, codificacion de etiquetas, división de datos en conjunto de e
data = pd.read_csv('evaluacion.csv')
print(data.columns)
data.head()
```

```
Index(['filename', 'chroma_stft', 'rmse', 'spectral_centroid',
       'spectral_bandwidth', 'rolloff', 'zero_crossing_rate', 'mfcc1', 'mfcc2',
       'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10',
       'mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17',
       'mfcc18', 'mfcc19', 'mfcc20'],
      dtype='object')
```

	filename	chroma_stft	rmse	spectral_centroid	spectral_bandwidth	rolloff	zero_crossing_rate	mfcc1	mfcc2	mfcc3
0	./Test/grabacion3.wav	0.238761	0.100748	1303.748788	1452.442740	2589.743255	0.056743	-289.824524	136.910706	-10.651265
1	./Test/grabacion1.wav	0.300142	0.024742	2370.466372	2371.520832	4814.628462	0.093106	-305.205841	82.311226	-18.697330
2	./Test/grabacion2.wav	0.272804	0.034967	2157.752902	2197.483927	3800.109579	0.122979	-301.068481	117.338799	-3.485050

3 rows × 27 columns

```
<Figure size 576x576 with 0 Axes>
```

Fig 23.

Luego validamos con data real y tenemos el siguiente resultado:

```
# Procesamiento de datos: Cargar la data del CSV, codificacion de etiquetas, división de datos en conjunto de
data = pd.read_csv('evaluacion.csv')

# Dropping unneccesary columns
data = data.drop(['filename'],axis=1)

#Scaling the Feature columns
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype = float))

print("Evaluacion del Modelo con la data de prueba: ")
print("============================================ \n")
for i in range(0,len(X)):
    print('Prediction %s Probability: %.2f%%' %\
          (label[clf.predict(X)[i]], clf.predict_proba(X)[i].max()*100))
```

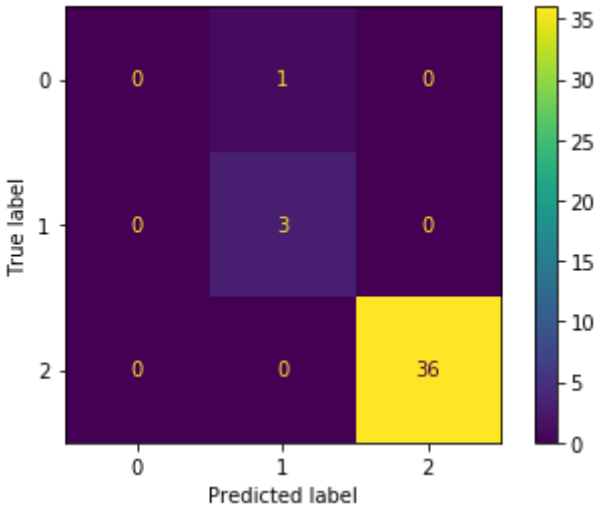
```
Evaluacion del Modelo con la data de prueba:
============================================

Prediction rango2 Probability: 100.00%
Prediction rango3 Probability: 100.00%
Prediction rango1 Probability: 87.87%
```

Fig 24.

VIII. RESULTADOS

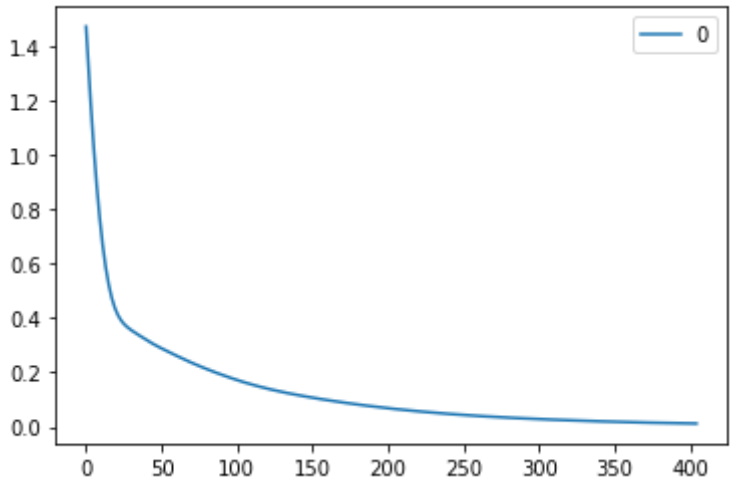
Los resultados encontrados inicialmente, con la data original se encontró que la Clase 2 es la que se tenía más información haciendo que el modelo aprenda más este patrón:



Sin embargo, se encontró que el número de iteraciones que se usó fue 405 menos del que se configuró (max_iter=500) y el accuracy era de 98.7%:

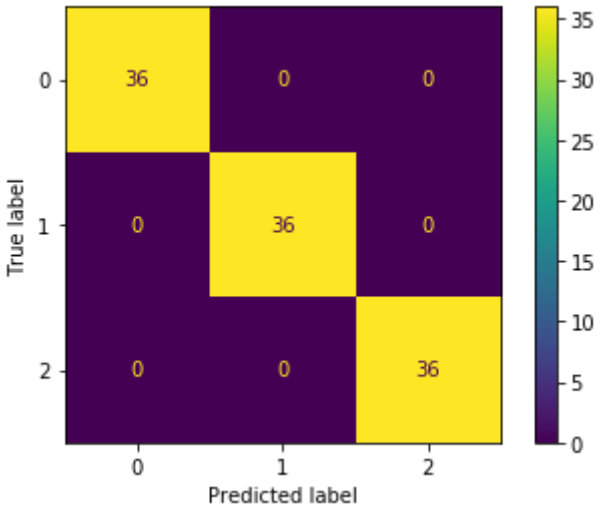
- Score: 0.975
- Acurracy: 97.5
- Recall: [0. 1. 1.]
- Precision: [0. 75. 100.]
- Funcion de acctivacion en la última capa: softmax
- Perdida obtenida durante el entranamiento: 0.02892402351832123
- Numero iteraciones: 405
- Numero de capas: [2, 26, 100]

Curva de pérdida:

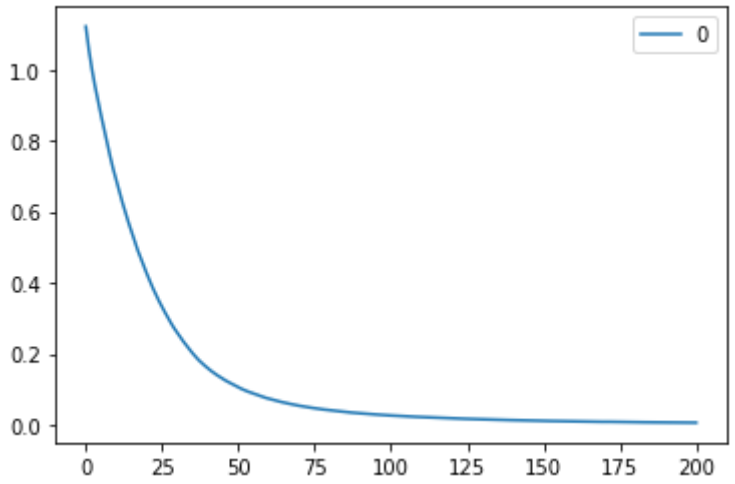


También se encontró que el número de iteraciones que se usó fue 201 menos del que se configuró (max_iter=300) y el accuracy era de 100%:

- `Score:  1.0`
- `Acurracy:  100.0`
- `Recall:  [1. 1. 1.]`
- `Precision:  [100. 100. 100.]`
- `Función de activación en la última capa:  softmax`
- `Pérdida obtenida durante el entranamiento: 0.005999909118060563`
- `Número iteraciones: 201`
- `Número de capas: [2, 26, 100]`



Y la curva de error es la siguiente:



IX. CONCLUSIONES

1. Las técnicas de inteligencia artificial son útiles para poder predecir el rango de edad del hablante. Esta característica puede ser usada como filtro adicional de identificación para validar la identidad del hablante cuando se requiera al realizar transacciones no presenciales.
2. También podrían ser usadas con éxito como parte de los sistemas operativos que intentar identificar, usando hardware, a la persona que interactúa con ellas.
3. Otra aplicación útil podría ser el bloqueo de accesos a contenido en internet que no corresponda con la edad del solicitante.
4. También podría usarse para las actividades de marketing para la recomendación de productos adecuados para una determinada edad.

X. RECOMENDACIONES

XI. REFERENCIAS

[1] S. Zhou, A Transfer Learning Method for Speech Emotion Recognition from Automatic Speech Recognition, 2020.

[2] G. Martínez, Reconocimiento de voz basado en MFCC, SBC y Espectrogramas, 2013.

[3] H. Gamper, Sound Sample Detection and Numerosity Estimation Using Auditory Display, 2013.

[4] J. Villena, Diseño e Implementación de un Sistema de Reconocimiento de Hablantes, 2014.

[5] T. Giannakopoulos, Introduction to Audio Analysis, 2014.

[6] Speech-Based Emotion Recognition using Neural Networks and Information Visualization

[7] A Transfer Learning Method for Speech Emotion Recognition from Automatic Speech Recognition

Anexo 1: CÓDIGO FUENTE

```
#################################################################
###################### Librerias usadas  ########################
from sklearn.neural_network import MLPClassifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import librosa
from pydub import AudioSegment
import csv
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
import pickle
#################################################################



#################################################################
################### Preparacion de Datos  ###################

# Crear expectogramas a partir de los audios.
# 1. Convertir los audios.wav en png para poder crear los atributos.

# Definimos el color map que usaremos para crear el espectograma
cmap = plt.get_cmap('inferno')
plt.figure(figsize=(8,8))

# ./TestData es la ruta donde esta todos los audios etiquetados Audio00x_xyz_sloganw.
path = r'./RawData'
extension = '.wav'


for root, dirs_list, files_list in os.walk(path):
        for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
        file_name_path = os.path.join(root, file_name)
        name=os.path.splitext(file_name)[0]
        y, sr = librosa.load(file_name_path, mono=True, duration=5)
        plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='default', mode='default', scale='dB');
        plt.axis('off');
        plt.savefig(f'{file_name_path[:-3]}png')
        plt.clf()
#################################################################



#################################################################
################### Creacion del dataset  ###################

# Funcion para hacer el mapping de los archivos de audio con la clase
def identificar_parametros(filename):
        cadena1 = file_name
        if cadena1.find("otros")>=0 :
        return "otros"
        if cadena1.find("rango1")>=0 :
        return "rango1"
        if cadena1.find("rango2")>=0 :
```

```
        return "rango2"
        if cadena1.find("rango3")>=0 :
        return "rango3"
        else:
        return 999

# Creamos la lista de clases
slogan = 'rango1 rango2 rango3 otros'.split()

# Creamos la cabecera del data set
header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth rolloff zero_crossing_rate'
for i in range(1, 21):
        header += f' mfcc{i}'
header += ' clase'
header = header.split()

# Creamos el archivo dataset.csv que tiene como primera columna la variale header
file = open('dataset.csv', 'w', newline='')
with file:
        writer = csv.writer(file)
        writer.writerow(header)

# Extraemos las caracteristicas desde el Espectograma que se convertiran en atributos del dataset:
# 1. Mel-frequency cepstral coefficients (MFCC)
# 2. Spectral Centroid
# 3. Zero Crossing Rate
# 4. Chroma Frequencies
# 5. Spectral Roll-off.
path = r'./RawData'
extension = '.wav'
for root, dirs_list, files_list in os.walk(path):
        for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
        file_name_path = os.path.join(root, file_name)
        audio = file_name_path
        y, sr = librosa.load(audio, mono=True, duration=30)
        rmse = librosa.feature.rms(y=y)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        p_slogan = identificar_parametros(file_name)
        to_append = f'{file_name_path} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np.mean(spec_bw)}
{np.mean(rolloff)} {np.mean(zcr)}'
        for e in mfcc:
        to_append += f' {np.mean(e)}'
        to_append += f' {p_slogan}'
        file = open('dataset.csv', 'a', newline='')
        with file:
        writer = csv.writer(file)
        writer.writerow(to_append.split())

# Cargamos y leemos los datos en la variable pandas data.
# Contiene las etiquetas y atributos.
data = pd.read_csv('dataset.csv')
data.head(10)



########################################################################################################
############## Preparacion, Entendimiento, Limpieza y Normalización del dataset  ###################
```

```
###### 1. NULLs or NA #######
# Verificamos si existen datos Null, NA en el dataset.
data = pd.read_csv('dataset.csv')
print("Datos Null y NA en el dataset")
print("============================== \n")
print(data.isnull().sum())
print("\n")
print("Datos Null en todo el dataset: ", data.isnull().values.any())
print("Datos NA en todo el dataset: ", data.isna().sum().sum())

###### 2. Outlayers or Datos extremos #######
# Verificamos si existen datos extremos en el dataset.
data = pd.read_csv('dataset.csv')

print(data["clase"].value_counts())

print("Asimetria de los datos:\n", data.skew())

#Filtramos las columanas para poder analizar por grupos
columns = data.columns
print(data.columns)
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
cols = [c for c in data.columns if data[c].dtype in numerics]

cols1 = ['chroma_stft', 'rmse', 'spectral_centroid', 'spectral_bandwidth', 'rolloff', 'zero_crossing_rate']
cols2 = ['mfcc1', 'mfcc2', 'mfcc3', 'mfcc4', 'mfcc5', 'mfcc6', 'mfcc7', 'mfcc8', 'mfcc9', 'mfcc10']
cols3 = ['mfcc11', 'mfcc12', 'mfcc13', 'mfcc14', 'mfcc15', 'mfcc16', 'mfcc17', 'mfcc18', 'mfcc19', 'mfcc20']

#Generamos 3 tipos de dataset para entender la dispercion que tienen los datos
data1 = data[cols1]
data2 = data[cols2]
data3 = data[cols3]

ax1 = data1.hist(bins= 10, alpha=0.5)
ax1 = data2.hist(bins= 10, alpha=0.5)
ax1 = data3.hist(bins= 10, alpha=0.5)
plt.show()

# Calculamos los quartiles y el IQR
Q0 = data1.quantile(0.00)
Q1 = data1.quantile(0.25)
Q2 = data1.quantile(0.50)
Q3 = data1.quantile(0.75)
Q4 = data1.quantile(1.00)

# Creamos el IRQ con el 3er y 1er quartil
IQR= Q3 - Q1

# Creamos el filtro que nos validara la existencia de valores extremos que este fuera del 1.5*IRQ en cada columna
filtro = ((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)
outliers = data[filtro]

# Sacamos los datos extemos y actualizamos el dataset
data_new = data[~filtro]

# Actualizamos dataset
data = data_new

#Grafico de los datos extremos
plt.figure(figsize=(20,20))
outliers.boxplot()
plt.show()

#Grafica sin los datos extremos
```

```
plt.figure(figsize=(20,20))
data_new.boxplot()
plt.show()

data.head()

###### 3. Balanceo de datos mediante las clases #######
# Primero verificamos que una gran diferencia entre la clase "otros" y las otras clases slogan1,2,3,4,5...

# Buscamos los valores de cada clase en base a filas
rango3, rango2, rango1 = data["clase"].value_counts()

# Según esto podemos verificar que existe un gran desbalance entre la clase "otros" y las otras clases
# Cantidad de filas por tipo de clase
print("Cantidad de filas por Clase antes del balanceo: \n", data["clase"].value_counts())

#Creamos un filtro para cada clase
g1 = data.loc[data["clase"]=="rango1"]
g2 = data.loc[data["clase"]=="rango2"]
g3 = data.loc[data["clase"]=="rango3"]

# Luego aplicamos la tecnica de Over-sample, que es utilizada para rellenar a las cantidades minoritarias
# que en este caso son las clases slogan1, slogan2, ... para que se iguale en cantidad con la clase otros.
g1_over = g1.sample(rango3, replace=True)
g2_over = g2.sample(rango3, replace=True)
g3_over = g3.sample(rango3, replace=True)


# Balanceamos los datos
data_balanced = pd.concat([g1_over, g2_over, g3_over])
# Volvemos a verificar el desbalance entre la clase "otros" y las otras clases, y confirmamos que ya no existe
# el desbalance.
# Cantidad de filas por tipo de clase luego del balanceo
print("\n")
print("Cantidad de filas por Clase despues del balanceo: \n", data_balanced['clase'].value_counts())
print(data_balanced.shape)
data_balanced.head(10)

# Actualizamos dataset
data = data_balanced

###### 4. Eliminación de columnas innesesarias #######
# Eliminamos las columanas que no utilizamos como la columna "filename" que es usada como nombre del archivo.
data = data.drop(['filename'],axis=1)
data.head(10)

###### 5. Normalización #######

X = data.iloc[:, :-1]
print(np.array(data.iloc[:, :-1]))

minmax_scale = preprocessing.MinMaxScaler().fit(X)
X = minmax_scale.transform(np.array(data.iloc[:, :-1], dtype = float))

############ 6. Codificación de la clase ##############

clase_list = data.iloc[:,-1]
encoder = LabelEncoder()
y = encoder.fit_transform(clase_list)
data["clase"] = y
print(y)

data.head(10)
```

```
#################################################################
######### Creación del Modelo y Entrenamiento ###############
#
# Para crear el modelo usaremos la tecnica Stratified K-fold (version mejorada de la tecnica K-fold)
#para separar con la data de prueba y de entrenamiento

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

def isnp(M):
   if isinstance(M, np.ndarray): return True
   return False

def dim(M):
            if isnp(M): M = list(M)

            if not type(M) == list:
            return []

            return [len(M)] + dim(M[0])


# Creacion del modelo con el algoritmo MLPClassifier ANN con las siguientes caracteristicas
# 1. Uso funcion de activacion softmax en la ultima capa(Esta configuracion esta en la logica del algoritmo)
# 2. Uso maximo de 300 iteracion
# 3. Uso de 3 capas
clf = MLPClassifier(random_state=1, max_iter=300)

#######
# separa los datos para entrenamiento y para pruebas
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=1, test_size=0.2)
#print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# entrena clasificador
clf = MLPClassifier(random_state=1, max_iter=500).fit(X_train,y_train)
print(clf)

# calcula probabilidades, solo el primero
y_test_prob = clf.predict_proba(X_test)
#print('X_test', X_test)
#print('y_test_prob', y_test_prob)

# predice resultados, solo el primero
y_test_pred = clf.predict(X_test)
#print('X_test', X_test)
#print('y_test_prd', y_test_pred)

# matriz de confusion
cmatrix = confusion_matrix(y_test, y_test_pred)
print(cmatrix)
plot_confusion_matrix(clf, X_test, y_test)
plt.show()

# Evaluacion del modelo
print("Score: ",  clf.score(X_test, y_test))
print("Acurracy: ", accuracy_score(y_test,y_test_pred)*100)
print("Recall: ", recall_score(y_test, y_test_pred, average=None))
print("Precision: ", precision_score(y_test, y_test_pred, average=None)*100)

print("Funcion de acctivacion en la ultima capa: ", clf.out_activation_)
print("Perdida obtenida durante el entranamiento")
print(clf.loss_)
print(clf.best_loss_)
print("Numero iteraciones")
```

```
print(clf.n_iter_)
print("Numero de capas")
print(dim(clf.coefs_))

#######

pd.DataFrame(clf.loss_curve_).plot()
plt.show()


################################################################
################### Evaluación del Modelo ###################

# Creacion del clasificador
pickle.dump(clf, open('classifier.pkl', 'wb'), protocol=4)

# Cargamos el modelo para su evaluacion
clf = pickle.load(open('classifier.pkl', 'rb'))


label = {0: 'rango1',
        1: 'rango2',
        2: 'rango3'
        }

# Creacion de dataset(evaluación.csv) sin clase el cual se usará pra evaluar el modelo.

cmap = plt.get_cmap('inferno')
plt.figure(figsize=(8,8))

path = r'./Test'
extension = '.wav'
for root, dirs_list, files_list in os.walk(path):
        for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
        file_name_path = os.path.join(root, file_name)
        name=os.path.splitext(file_name)[0]
        y, sr = librosa.load(file_name_path, mono=True, duration=5)
        plt.specgram(y, NFFT=2048, Fs=2, Fc=0, noverlap=128, cmap=cmap, sides='default', mode='default', scale='dB');
        plt.axis('off');
        plt.savefig(f'{file_name_path[:-3]}png')
        plt.clf()

header = 'filename chroma_stft rmse spectral_centroid spectral_bandwidth rolloff zero_crossing_rate'
for i in range(1, 21):
        header += f' mfcc{i}'
header = header.split()

# Extraer caracteristicas desde el Espectograma: Mel-frequency cepstral coefficients (MFCC), Spectral Centroid, Zero Crossing Rate,
Chroma Frequencies, y Spectral Roll-off.
file = open('evaluacion.csv', 'w', newline='')
with file:
        writer = csv.writer(file)
        writer.writerow(header)

extension = '.wav'
for root, dirs_list, files_list in os.walk(path):
        for file_name in files_list:
        if os.path.splitext(file_name)[-1] == extension:
        file_name_path = os.path.join(root, file_name)
        audio = file_name_path
        y, sr = librosa.load(audio, mono=True, duration=30)
        rmse = librosa.feature.rms(y=y)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
```

```
        spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
        spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
        rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
        zcr = librosa.feature.zero_crossing_rate(y)
        mfcc = librosa.feature.mfcc(y=y, sr=sr)
        to_append = f'{file_name_path} {np.mean(chroma_stft)} {np.mean(rmse)} {np.mean(spec_cent)} {np.mean(spec_bw)}
{np.mean(rolloff)} {np.mean(zcr)}'
        for e in mfcc:
        to_append += f' {np.mean(e)}'
        file = open('evaluacion.csv', 'a', newline='')
        with file:
        writer = csv.writer(file)
        writer.writerow(to_append.split())

# Procesamiento de datos: Cargar la data del CSV, codificación de etiquetas, división de datos en conjunto de entrenamiento y
#prueba.
data = pd.read_csv('evaluacion.csv')
print(data.columns)
data.head()

# Dropping unneccesary columns
data = data.drop(['filename'],axis=1)

#Scaling the Feature columns
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data.iloc[:, :-1], dtype = float))

print("Evaluacion del Modelo con la data de prueba: ")
print("============================================ \n")
for i in range(0,len(X)):
        print('Prediction %s Probability: %.2f%%' %       (label[clf.predict(X)[i]], clf.predict_proba(X)[i].max()*100))
```