Tópicos de Ingeniería Mecatrónica
MT0017

Proyecto Parcial

Algoritmo de Aprendizaje para la clasificación musical mediante Deep Learning

Alejandro Muñoz Zevallos

Aaron Figueroa Bianchi

Piero Zela Marcos

2020-2

Índice General

1. MARCO CONCEPTUAL

1.1. Introducción

En los últimos años la demanda en la industria musical ha tenido un crecimiento exponencial producto del fenómeno de distribución musical a través de internet, esto ha ocasionado un aumento en el número de géneros musicales, artistas, instrumentos, estructuras y la base de datos relacionado a este entorno. Es por ello que se ha realizado diversos estudios sobre qué características o parámetros son relevantes para hacer clasificación y qué técnicas de aprendizaje computacional son efectivas para procesar esta información. Algunos estudios relevantes son: Seth Golub realizó una investigación aplicando 3 diferentes tipos de clasificadores a dos géneros musicales, Paul Scott elaboró un clasificador con perceptrón multinivel para cuatro géneros musicales; y Soltau entrenó una red neuronal auto-asociativa para rock, tecno, pop y clásico, todos estos estudios enfocados a Machine Learning.

Estos estudios incentivan a la industria a realizar más investigación sobre cómo manejar esta gran base datos y cómo poder clasificarlas.

1.2. Objetivos y resultados esperados

1.2.1. Objetivo General

Desarrollar un algoritmo de aprendizaje capaz de clasificar el carácter de una canción.

1.2.2. Objetivos Específicos

Implementar una base de datos de aspecto musical

Aplicar teoría de Machine Learning

Construcción de un modelo computacional para un bajo costo.

1.2.3. Resultados esperados

Se espera que el algoritmo pueda identificar los 2 carácter musicales de una canción con una exactitud mayor a 85%.

2. PROCEDIMIENTO

2.1. Según sus géneros musicales y sus características

A continuación se muestra el número de canciones según su carácter o estado de ánimo (bajo o alto) y según su género musical. Estas canciones fueron usadas para entrenar y testear los diferentes métodos de clasificación que elaboramos:

CLASE	GÉNERO MUSICAL	CARÁCTER		POBLACIÓN	
		ALTO	BAJO		
1	Blues	44	56	100	
2	Classical	10	90	100	
3	Country	33	67	100	
4	Disco	94	6	100	
5	Hip-hop	70	30	100	
6	Jazz	22	77	99	
7	Metal	96	4	100	
8	Pop	58	42	100	
9	Reggae	9	91	100	
10	Rock	65	35	100	
		501	498	999	TOTAL

Figura 1. Géneros musicales y sus estados de ánimo altos y bajos

2.2. Métodos propuestos

Se realizarán múltiples pruebas con los diferentes modelos vistos en clase y se compara su accuracy score usando las librerías de sklearn. Además, se grafica la matriz de confusión de cada uno de los experimentos y se explicará el porqué de los resultados obtenidos mediante la teoría. Para ello se utiliza el 80 % de los datos para entrenamiento y el 20 % de los datos para testeo, de esta manera podemos estar seguro de que nuestro modelo presenta suficientes datos para no generalizar un modelo y presentar overfitting.

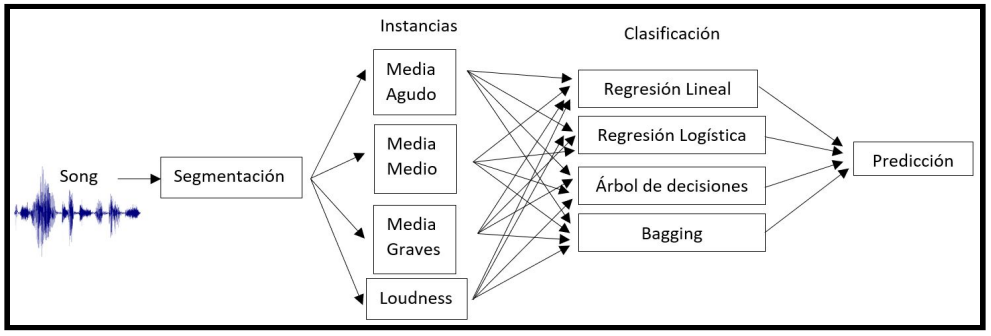


Figura 2. Modelo de procesamiento de la señal

Figura 3. Separación de datos para el entrenamiento y el testeo

3. IMPLEMENTACIÓN DE ALGORITMOS DE MACHINE LEARNING

3.1. Métodos en Matlab

3.1.1. Clasificación por método de regresión logística

Algo que puede resultar de fácil obtención a la hora de clasificar canciones según el estado de ánimo que provoca es la distribución de ganancias a lo largo del espectro de frecuencias.
Las canciones animadas tienen más presencia de armónicos agudos que las canciones tranquilas. Esto se debe, en gran medida, a la distorsión (generalmente intencional) que se genera al momento de recortar la señal cuando esta llega a los 0dB al ser procesadas por las tiendas digitales de música (o -1dB, según sea el caso).

Tomemos como ejemplo la siguiente señal compuesta de 3 ondas senoidales puras de aproximadamente la misma amplitud (la señal está enventanada).

Figura 4. Suma de 3 ondas senoidales

Podemos ver las amplitudes individuales de estas frecuencias en el siguiente analizador de frecuencias (un tipo de analizador que hace uso de la transformada de fourier).

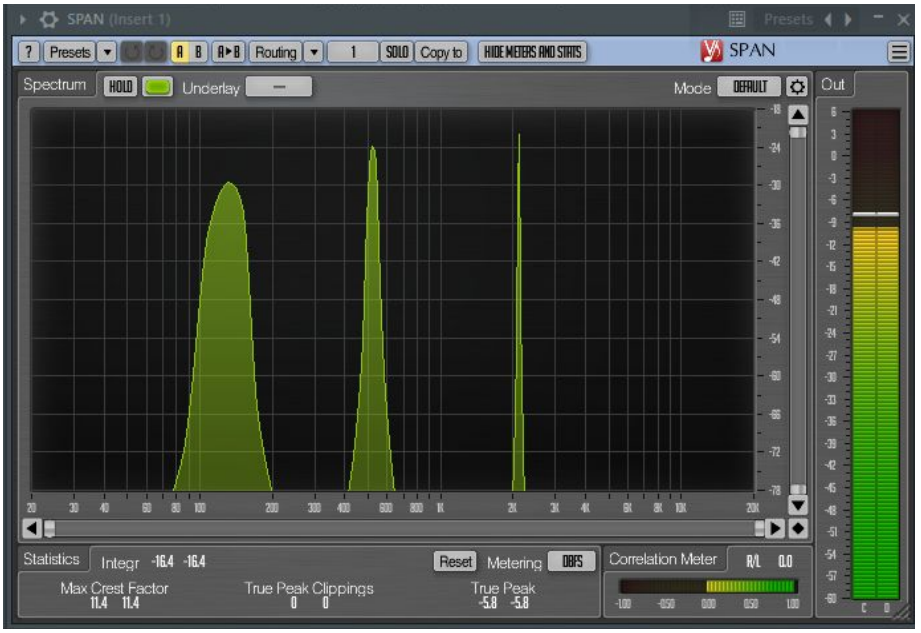


Figura 5. Espectrograma de las 3 ondas senoidales

Como se puede observar en el espectro, no hay mucha presencia de agudos en la señal obtenida, dado que solo tenemos 3 ondas senoidales.

- El rango de los graves parte en el inicio del rango de audición humana (20Hz aproximadamente) y termina aproximadamente en los 250Hz, los medios continúan hasta los 2000Hz aproximadamente y los agudos continúan hasta el fin del rango de audición humana (aproximadamente 18000Hz).

Figura 6. Un armónico en la parte de los agudos

Se puede notar en el siguiente analizador de ganancia que la señal no tiene la suficiente amplitud como para ser recortada:

- El eje horizontal de la gráfica indica la amplitud de la señal antes de ser recortada y el vertical representa la amplitud después del recorte.
- La línea blanca clara representa la relación entre ambos ejes
- La línea suave vertical indica el valor de la amplitud que entra en el analizador

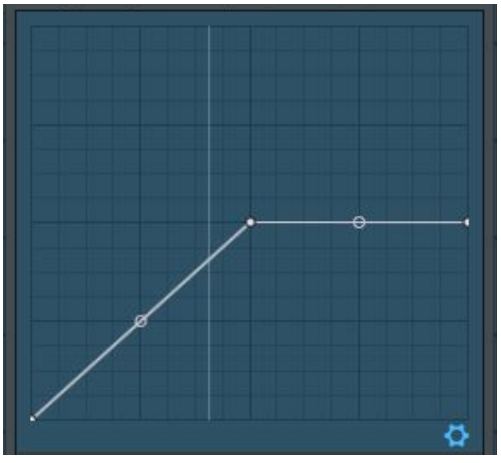


Figura 7. No es necesario recortar la señal, dado que no se llega a los 0dB

Sin embargo, si incrementamos la amplitud de la señal de entrada, (subimos el volumen), dado que los servicios de streaming siempre recortan la señal de entrada para que no supere (como máximo) los 0dB, la señal se distorsionara:

Figura 8. Señal ligeramente recortada. La línea roja indica el umbral del recorte.

Se puede ver que la señal original tuvo que ser recortada levemente en el gráfico anterior y en el analizador de ganancia:

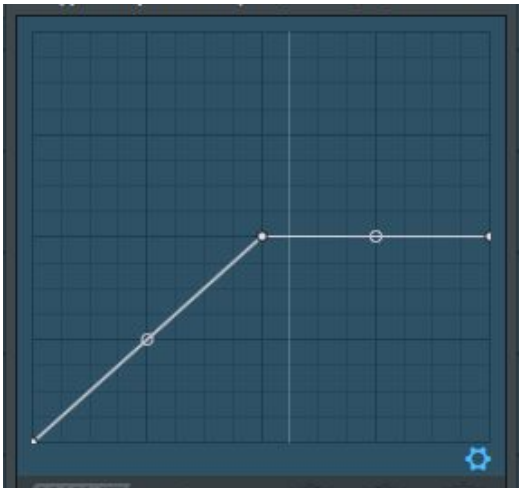


Figura 9. La señal sobrepasa ligeramente el umbral de recorte

Fruto de esto, se generan armónicos agudos que incrementan en gran medida la ganancia promedio de la sección de agudos de la señal. Estos armónicos se pueden visualizar en el analizador de frecuencias:

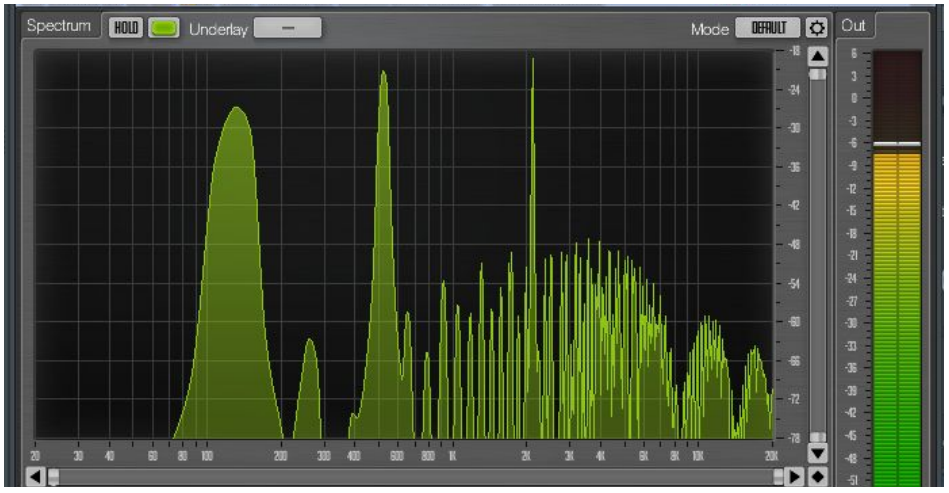


Figura 10. Armónicos agudos generados

Normalmente, este fenómeno se produce de manera intencional en los casos de las guitarras eléctricas y las baterías distorsionadas con el objetivo de que sean más notorias sin pasar realmente de los 0dB

Sin embargo, este no es el único fenómeno que podemos aprovechar de la saturación.

Como hemos visto, mientras más se satura un onda, más se va pareciendo a una onda cuadrada, lo que hace que su sonoridad promedio incremente.

Figura 10.1. Onda senoidal sin saturar

Figura 10.2. Onda senoidal muy saturada

- Sonoridad promedio o “mean loudness” es un término usado para describir cuán ruidosa es una canción. Se obtiene simplemente hallando el valor promedio del valor absoluto de la señal a analizar.

Se puede observar en la siguiente gráfica que el valor promedio de una onda senoidal de amplitud 1 es de aproximadamente 0.637:

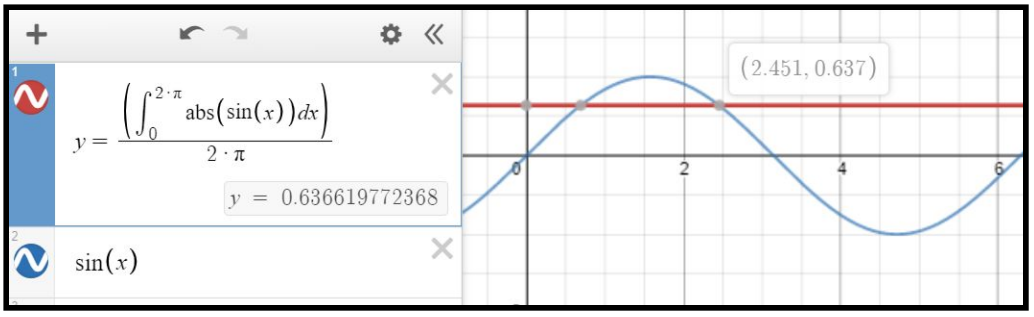


Figura 11.1 valor promedio de onda senoidal

Por otro lado, el valor promedio de una onda cuadrada de amplitud 1 es 1:

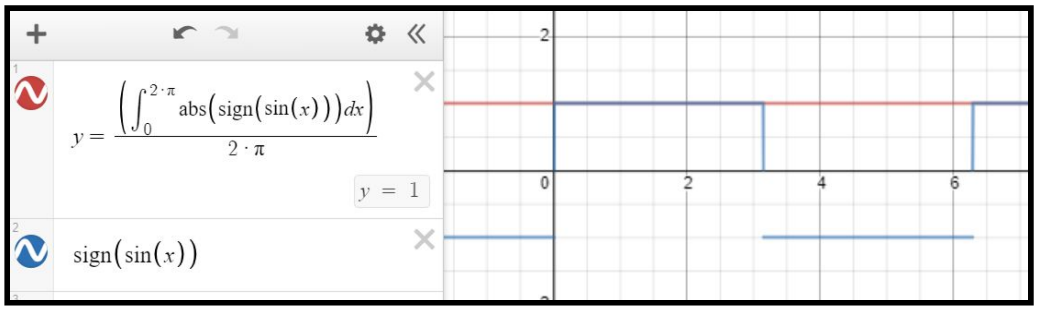


Figura 11.2 valor promedio de onda cuadrada

Por lo que podemos concluir que, mientras más se sature una onda, más ruidosa será aparentemente sin importar que comparemos señales que, técnicamente, tienen la misma amplitud máxima.

Adicionalmente, podemos aprovechar el hecho de que los géneros más violentos o animados como el Rock, Metal o Electrónica se caracterizan por tener bajos y agudos bastante pronunciados. Como resultado, la ganancia en los medios se ve mermada dado que es necesario cumplir unos estándares de sonoridad para la gran mayoría de las distribuidoras digitales. Por otro lado, los géneros más calmados como la música clásica, jazz o reggae no tienen tanta presencia en los graves ni agudos, ya que es difícil que un instrumento llegue naturalmente a frecuencias demasiado bajas o elevadas sin el uso de bajos electrónicos o de efectos como la distorsión.

Figura 12.1. Espectro de frecuencias de una canción ambiental (Pocos graves y mayoría de medios)

Figura 12.2. Espectro de frecuencias de una canción del género Metal (Graves y agudos abundantes. Medios moderados)

Por último, cabe mencionar que, si bien es cierto que se puede medir cuán animada es una canción simplemente viendo cual de ellas presenta un pico de amplitud más alto, este método podría ser falseado simplemente bajando el volumen de una canción animada y comparándola con una tranquila a la que se le ha subido el volumen. Por esto, es necesario normalizar todas las canciones antes de que sean analizadas. De esta manera, se podrá determinar cuán animada o tranquila será por mucho que se le haya subido o bajado el volumen.

Figura 13. Comparativa entre canción animada (abajo) y canción ambiental (arriba). Ambas con la ganancia alterada artificialmente.

También es importante señalar que el tempo de una canción, aparte de ser muy difícil de obtener de manera automática y rápida, no ayuda en gran medida en saber el ánimo de una canción, ya que depende de cómo se usen las subdivisiones del tempo para aumentar o disminuir la sensación de tranquilidad.

- Tempo o BPM (Beats per minute) es la medida que describe cuántos pulsos por minuto posee una canción.

Como se puede apreciar en la siguiente imagen, ambos loops de batería tienen el mismo tempo, pero el de abajo es mucho más animado que el de arriba porque hace mucho más uso de subdivisiones. Los marcadores rojos representan el inicio de cada pulso.

Figura 14. Comparación de loops de batería con el mismo tempo

En resumen, los atributos que usaremos para determinar si una canción es animada o tranquila serán, en mayor medida, el mean loudness y la presencia de agudos y, en menor medida, la presencia de graves y medios.

3.1.2. Entrenamiento de la regresión logística en Matlab

Con el objetivo de simplificar la obtención de parámetros de cada instancia que usaremos para entrenar la regresión de clasificación, escribimos un programa/función en matlab llamado “atrib” cuya entrada es la señal de la canción y su frecuencia de muestreo. Este programa determinará automáticamente los 4 atributos necesarios que se mencionaron anteriormente.

```
function [mean_low,mean_mid,mean_high,mean_loudness]=atrib(ys,fs)
```

Para determinar el mean loudness, usamos la función mean, que hace un promedio de todos los valores de un vector, pero primero debemos normalizar la señal, por lo que dividiremos todo el vector entre su valor máximo. También convertimos la señal en mono en caso la canción tenga información estéreo con el objetivo de solo hacer un procesamiento por canción.

```
    if size(ys,2)==2
        y=(ys(:,1)+ys(:,2))/2;
    else
        y=ys;
    end

    y=y/max(y);
```

```
    mean_loudness=mean(abs(y));
```

El siguiente paso es obtener la amplitud promedio de cada banda de frecuencias. En un inicio, se eligieron las frecuencias entre bandas como 250 y 2000. Sin embargo, posteriormente se cambiaron a 200 y 3000 dado que entregaban un mejor resultado en el programa final.

```
    low_start=20;
    low_mid=300;
    mid_high=2000;
    %high_end=18000;

    yfft=abs(fft(y));
    %yfft=periodogram(y);
    yfft=yfft(1:fix(length(yfft)/2),:);
    %yfft=yfft/max(yfft);
    %freq=linspace(0,fs/2,length(yfft));
    %figure,plot(freq,yfft),hold on


yfft_low=yfft(fix(length(yfft)*low_start/(fs/2)):fix(length(yfft)*low_mid/
(fs/2)));

%freq_low=freq(fix(length(yfft)*low_start/(fs/2)):fix(length(freq)*low_sta
rt/(fs/2)));


yfft_mid=yfft(fix(length(yfft)*low_mid/(fs/2)):fix(length(yfft)*mid_high/(
fs/2)));

%freq_mid=freq(fix(length(freq)*low_mid/(fs/2)):fix(length(freq)*mid_high/
(fs/2)));

    yfft_high=yfft(fix(length(yfft)*mid_high/(fs/2)):end);
    %freq_high=freq(fix(length(freq)*mid_high/(fs/2)):end);

    %plot(freq_low,yfft_low,'r')
    %plot(freq_mid,yfft_mid,'k')
    %plot(freq_high,yfft_high,'b')
    %hold off,

    %%%%%%%%%%%%%%%%
    mean_low=mean(yfft_low);
    mean_mid=mean(yfft_mid);
    mean_high=mean(yfft_high);

    mean_loudness=mean(abs(y));

end
```

De esta manera, podemos separar los valores promedio de las diferentes bandas que componen cada canción.

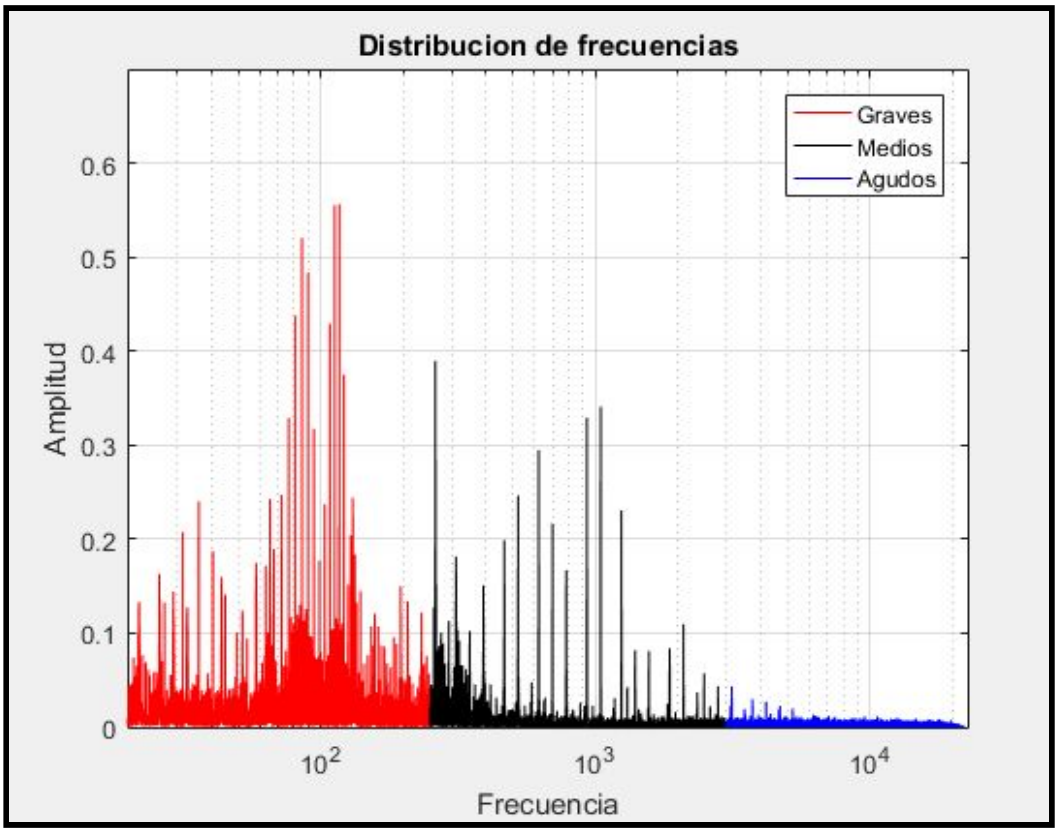


Figura 15. Distribución de frecuencias de una canción

Una vez creada esta función, podemos extraer los atributos de cada canción que queramos automáticamente. Para entrenar nuestra regla de clasificación, se usaron 813 fragmentos de 30 segundos de canciones de 10 géneros musicales distintos. Otros 206 fragmentos fueron usados posteriormente para validar la regresión obtenida.

```
clc, clear all, close all
%%  EXTRAER ATRIBUTOS
files=dir('C:\Users\dospu\Documents\MATLAB\topicos\Proyecto
1\songs_train');
% INICIALIZAMOS VECTORES

low=zeros(length(files)-2,1);
mid=zeros(length(files)-2,1);
high=zeros(length(files)-2,1);
loudness=zeros(length(files)-2,1);
for k=3:length(files)
    n=k-2;
    cd songs_train
    name=files(k).name;
    %names(n)=name
    [ys,fs]=audioread(files(k).name);
    cd ..
    [low(n),mid(n),high(n),loudness(n)]=atrib(ys,fs);
    %atributos(n,:)=[low(n),mid(n),high(n),loudness(n)];
end
x=[low, mid, high, loudness];
```

Una vez extraemos los atributos, procedemos a formar el clasificador.
Normalizamos los atributos obtenidos y añadimos una columna de unos a la izquierda para que acompañe al offset w0. El archivo “salida2.mat” es la respuesta de si una canción es animada (1) o calmada (0) y fue determinada manualmente.

Cabe destacar que, incluso para nosotros, fue difícil determinar si ciertas canciones eran calmadas o animadas:

```
%% FORMAR LA REGLA DE REGRESION
load('salida2.mat')

atrb=size(x,2);

%NORMALIZACION
for i=1:atrb
    x(:,i)=(x(:,i)-mean(x(:,i)))/(max(x(:,i))-min(x(:,i)));
end
atrb=atrb+1;
x=[ones(size(x,1),1) x];

w=ones(1,atrb);
y=salida2;
```

De esta manera, podemos iterar para encontrar los valores de la función de costo vs iteración y los pesos w:

```
n=size(x,1); %NUMERO DE MUESTRAS

alfa=5;    %TAMANO DE PASO
n_iter=3000;   %NUMERO DE ITERACIONES

for i=1:n_iter
    g=w*x';
    h=1./(1+exp(-g));   %FUNCION DE HIPOTESIS
    for cw=1:atrb
        wn(cw)= w(cw)-alfa/n*((h'-y)'*x(:,cw));
        w(cw)=wn(cw);
    end
    J(i)=-1/n*(y'*log(h')+(1-y')*log(1-h'));      %FUNCION DE COSTO
end

save w.mat w

figure,plot(J,'b','linewidth',2),title('Decremento del costo vs
iteraciones')
xlabel('Iteracion'),ylabel('Costo'),grid

Numero_de_iteraciones = n_iter
Tamano_de_paso_entre_iteraciones = alfa
Costo_alcanzado_en_entrenamiento = J(end)
```

Podemos verificar el desempeno de la regresion segun la data de entrenamiento de la siguiente manera:

```
%% DESEMPENO CON LA DATA DE ENTRENAMIENTO
files_test=dir('C:\Users\dospu\Documents\MATLAB\topicos\Proyecto
```

```
1\songs_test');

for n=1:length(files)-2
    g=x(n,:)*w';
    h(n)=round(1./(1+exp(-g)));
end
comparacion_train=[salida2,h'];
acertados=sum(and(salida2,h'))+sum(and(1-salida2,1-h'));
acertadont=length(salida2)-acertados;
porcetanje_exactitud=acertados/length(salida2)
```

De la misma manera, podemos extraer los atributos de la data de testeo y probar la regresión obtenida anteriormente:

```
%% TESTEANDO (primero sacamos atributos y luego comparamos)
load('salida2_test.mat')
names=zeros(length(files_test)-2,1);
low=zeros(length(files_test)-2,1);
mid=zeros(length(files_test)-2,1);
high=zeros(length(files_test)-2,1);
loudness=zeros(length(files_test)-2,1);
for k=3:length(files_test)
    n=k-2;
    cd songs_test
    name=files_test(k).name;
    %names(n)=name
    [ys,fs]=audioread(files_test(k).name);
    cd ..
    [low(n),mid(n),high(n),loudness(n)]=atrib(ys,fs);
    %atributos(n,:)=[low(n),mid(n),high(n),loudness(n)];
end
x_test=[low, mid, high, loudness];

for i=1:size(x_test,2)

x_test(:,i)=(x_test(:,i)-mean(x_test(:,i)))/(max(x_test(:,i))-min(x_test(:
,i)));
end

x_test=[ones(size(x_test,1),1) x_test];

for n=1:length(files_test)-2
    g=x_test(n,:)*w';
    h2(n)=round(1./(1+exp(-g)));
end
comparacion_test=[salida2_test,h2'];

acertados_test=sum(and(salida2_test,h2'))+sum(and(1-salida2_test,1-h2'));
acertadont_test=length(salida2_test)-acertados_test;
porcetanje_exactitud_test=acertados_test/length(salida2_test)
```

Como se puede observar en el siguiente gráfico de Costo vs. Iteraciones, se alcanzó un costo mínimo de 0.4644

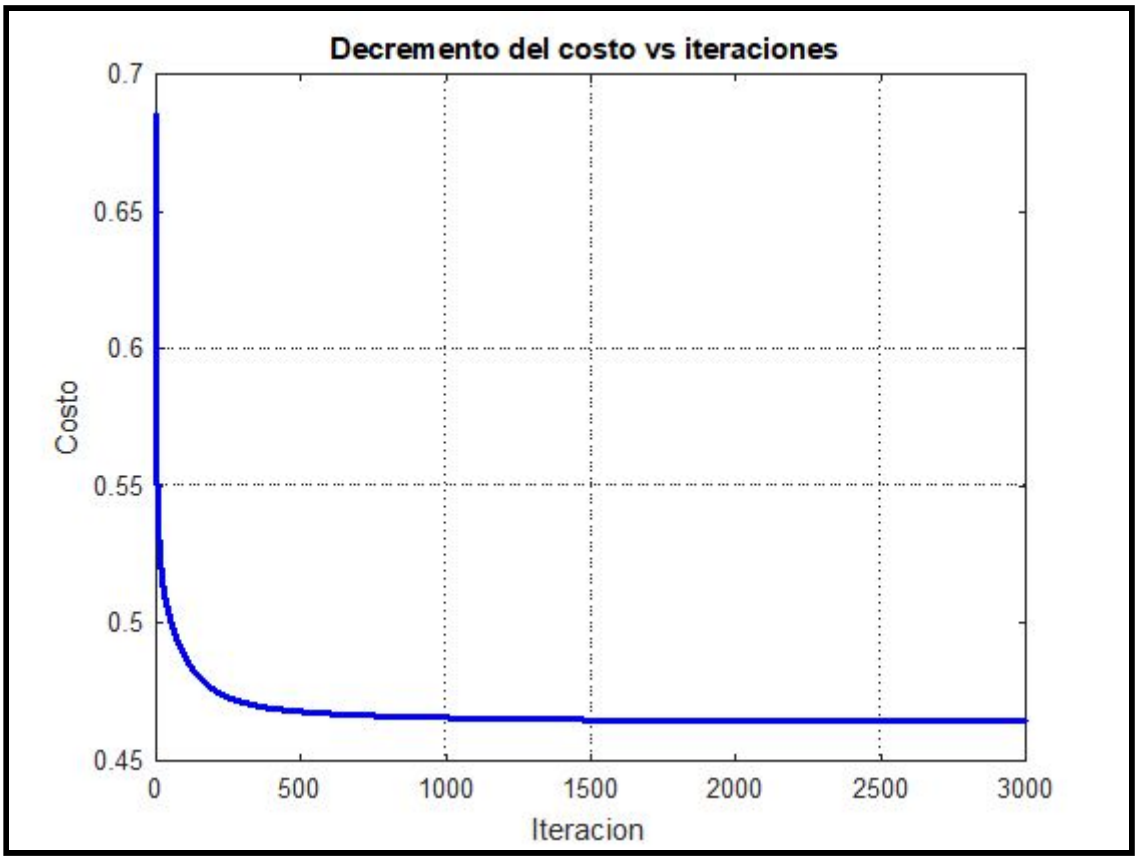


Figura 16. Decremento de la función de costo

Los resultados obtenidos fueron bastantes decentes (81% de accuracy):

```
Numero_de_iteraciones =

        3000


Tamano_de_paso_entre_iteraciones =

     5


Costo_alcanzado_en_entrenamiento =

    0.4644


porcetanje_exactitud =

    0.8106


porcetanje_exactitud_test =

    0.8107
```

Los pesos w obtenidos son los siguientes:

```
w = [0.1308    3.1553    2.6521   11.5634   -7.3670]
```

Es importante mencionar que estas ponderaciones fueron determinadas una vez se normalizó la data de entrenamiento. Si se deseara implementar esta regresión con una única canción individual, no se podría hacer ninguna normalización previa, dado que solo se tendría una instancia. En los casos en los que no se normalizaran los atributos antes de formar la regresión, las ponderaciones w de ese proceso serían las siguientes:

```
w = [-1126.7    -0525.8   -1124.1    9678.1   -0008.7]
```

Desafortunadamente, los porcentajes de accuracy en estos casos bajaría hasta el 59.04% para la data de entrenamiento y 55.83% para la data de testeo; por lo que, prácticamente, sería como lanzar una moneda al aire.

Sin embargo, podemos usar los datos estadísticos obtenidos para normalizar los atributos de entrenamiento con el objetivo de modificar los atributos de la canción individual que queramos analizar de la misma manera para que funcionen con las ponderaciones w halladas en un principio.

Para lograr esto, debemos guardar el promedio, valor máximo y mínimo de cada atributo al momento de normalizar:

```
%NORMALIZACION
for i=1:atrb
    norm_x(1,i)=mean(x(:,i));
    norm_x(2,i)=max(x(:,i));
    norm_x(3,i)=min(x(:,i));
    x(:,i)=(x(:,i)-norm_x(1,i))/(norm_x(2,i)-norm_x(3,i));
end
save norm_x.mat norm_x
```

Posteriormente, junto con las ponderaciones w, los podremos usar en una función de predicción para canciones individuales creada para agilizar el proceso de demostración del funcionamiento del programa:

```
function h=pred(cancion)
    load('w.mat'), load('norm_x.mat')
    [ys,fs]=audioread(cancion);

    [low, mid, high, loudness]=atrib(ys,fs);
    x=[low mid high loudness];
    for i=1:4
        x(i)=(x(i)-norm_x(1,i))/(norm_x(2,i)-norm_x(3,i));
    end
    x=[1 x];
    g=x*w';
    h=round(1/(1+exp(-g)));
end
```

Si se desea probar esta función, solamente sitúe una canción en la carpeta principal del proyecto y ejecute en el command window la siguiente línea de código. El

programa arrojará como resultado un 1 o un 0, dependiendo de si la canción ingresada es animada o calmada respectivamente:

```
>> pred('nombre.extension')
```

3.1.3. Resultados en de regresión logística en Matlab

Si probamos el programa de predicción individual con una canción calmada y, posteriormente, con otra animada, obtenemos:

```
>> pred('somthing like time_minecraft_2.mp3')

ans =

     0

>> pred('beam matrix.mp3')

ans =

     1
```

Figura 17. Demostración del correcto funcionamiento del programa con instancias individuales

Matriz de confusión (data de testeo):

		Valores predichos	
		1	0
Valores reales	1	78	22
	0	17	89

3.2. Método en Python Clasificación Binaria

3.2.1. Ordenamiento de los datos para el procesamiento en Python

Mediante programación en Python se realizó la transformada de Fourier de cada una de las canciones para obtener los bajos, medios, altos y el loudness. Se utilizó la librería ***"librosa"***, la cual permitió calcular características esenciales de una señal, tales como: transformada e inversa de Fourier, gráfica de espectrogramas, entre otras herramientas.

Las librerías importadas en ese proyecto son las mostradas en la Figura…..XX:

```
import os
import pandas as pd
import librosa
from librosa import display
import numpy as np
import IPython.display as ipd
from glob import glob
import matplotlib.pyplot as plt
%matplotlib inline
import random
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

Figura 18. Librerías usadas para clasificación binaria

La librería *glob* permitió colocar los directorios donde se encuentran los grupos de canciones por sus respectivos géneros:

```
train_blues = glob('{}/Proyecto/Songs/blues/*.wav'.format(os.path.abspath('../')
train_classical = glob('{}/Proyecto/Songs/classical/*.wav'.format(os.path.abspat
train_country = glob('{}/Proyecto/Songs/country/*.wav'.format(os.path.abspath('.
train_disco = glob('{}/Proyecto/Songs/disco/*.wav'.format(os.path.abspath('../')
train_jazz = glob('{}/Proyecto/Songs/jazz/*.wav'.format(os.path.abspath('../')))
train_rock = glob('{}/Proyecto/Songs/rock/*.wav'.format(os.path.abspath('../')))
train_hiphop = glob('{}/Proyecto/Songs/hiphop/*.wav'.format(os.path.abspath('../
train_metal = glob('{}/Proyecto/Songs/metal/*.wav'.format(os.path.abspath('../')
train_pop = glob('{}/Proyecto/Songs/pop/*.wav'.format(os.path.abspath('../')))
train_reggae = glob('{}/Proyecto/Songs/reggae/*.wav'.format(os.path.abspath('../
train_rock = glob('{}/Proyecto/Songs/rock/*.wav'.format(os.path.abspath('../')))
```

Figura 19. Asignación de clase para entrenamiento

Una vez subida las canciones,se procede a calcular la transformada de Fourier y se separa la banda de frecuencias de Fourier de 0 a 250 para graves, de 250 a 3000 para medios y de 3000 hasta todo el rango para agudos y calculamos la media de todas las frecuencias para cada una de las ‘i’ canciones de las 11 carpetas de géneros musicales.

Ya obtenido los promedios de los altos, medios, bajos y loudness de las canciones, se procede a realizar un *reshape* para tener un vector de 100 filas y una columna. Luego, usando *np.vstack* se procede a juntar todos los elementos de cada grupo en 4 diferentes columnas, para posteriormente utilizando *np.hstack* juntar todos los datos en un solo data frame llamado X.

Como respuesta tenemos la columna Y, la cual representa una salida binaria donde ‘1’ representa a una canción Movida y ‘0’ una canción Tranquila. Esta clasificación fue realizada con las 999 canciones del conjunto de datos. Esta columna será agregada al conjunto atributos utilizando np.hstack.

Para que el orden no sea un factor que afecte al entrenamiento del modelo, procedemos a realizar un muestreo sin reemplazo de un conjunto de números del 0 al 999 utilizando la función *random.sample()* para posteriormente utilizarlos como índices de nuestro conjunto de entrenamiento. De esta manera tendremos cada uno de los datos con su respectiva respuesta sin un orden por el género musical al que

pertenecen. Utilizamos np.random.seed para almacenar la semilla que nos dio los diferentes valores aleatorios y poder replicar múltiples veces el experimento con el mismo orden de valores.
Para el entrenamiento agarramos el 80% de los datos para entrenamiento y el otro 20% para testeo y los separamos en xtrain, ytrain, xtest, ytest.

3.2.2 Entrenamiento con múltiples métodos en Python

Una vez obtenidos los conjuntos de entrenamiento y de testeo, procedemos a aplicar diversos métodos de Machine Learning utilizando las librerías de sklearn y diversos conocimientos adquiridos en clase.

Empezaremos implementando una Regresión lineal de Ridge, para esto empezaremos utilizando la función GaussianFeatures vista en clase para generar una base gaussiana y usamos un alpha de 0.1:

```
In [20]: # Regresión Lineal
         from sklearn.linear_model import Ridge

         # Modelo de regresión lineal
         modeloRidge = make_pipeline(GaussianFeatures(30),Ridge(alpha=0.1))
         # Entrenamiento del modelo
         modeloRidge.fit(xtrain, ytrain)

Out[20]: Pipeline(memory=None,
              steps=[('gaussianfeatures', GaussianFeatures(N=30, width=None)), ('ridge',
         Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
            normalize=False, random_state=None, solver='auto', tol=0.001))])
```

Figura 20. Código para clasificación Ridge

Posteriormente, procedemos a predecir el comportamiento de xtest utilizando el modelo obtenido anteriormente.

```
ypredRidge= modeloRidge.predict(xtest)

ypredRidge=ypredRidge.reshape(199,)
```

Figura 21. Reshape a predicción mediante Ridge

Por último, se aproxima los valores del ypredRidge a 1 si son mayores iguales a 0.5 y a 0 si son menores a 0.5 utilizando la sentencia *ypredRidge.flatten()*

Calculamos el MSE y el accuracy_score del modelo y podemos ver que tenemos 2.216 y 75.37% respectivamente, por lo que podemos concluir que tenemos un buen modelo. A continuación se mostrará una tabla con varias métricas de evaluación

	precision	recall	f1-score	support
0.0	0.71	0.77	0.74	91
1.0	0.79	0.74	0.77	108
micro avg	0.75	0.75	0.75	199
macro avg	0.75	0.75	0.75	199
weighted avg	0.76	0.75	0.75	199

Figura 22. Resultado de métricas para Ridge

Usando confusion_matrix importado anteriormente de las métricas de Sklearn podemos graficar la matriz de confusión de nuestro modelo.

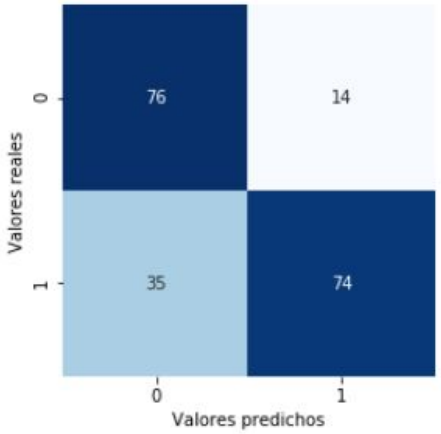


Figura 23. Matriz de confusión para Ridge

Luego probamos también el modelo de Lasso, importandolo desde las librerías de sklearn, usando una base gaussiana con 30 elementos y un alpha de 0.01

```
In [26]: # Regresión Lineal
         from sklearn.linear_model import Lasso

         # Modelo de regresión lineal
         modeloLasso = make_pipeline(GaussianFeatures(30),Lasso(alpha=0.01))
         # Entrenamiento del modelo
         modeloLasso.fit(xtrain, ytrain)

Out[26]: Pipeline(memory=None,
              steps=[('gaussianfeatures', GaussianFeatures(N=30, width=None)), ('lasso',
         Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
            normalize=False, positive=False, precompute=False, random_state=None,
            selection='cyclic', tol=0.0001, warm_start=False))])
```

Figura 24. Código para clasificación Lasso

Una vez entrenado el modelo, se obtuvo un MSE de 0.126 y un accuracy_score de 73.36%, lo cual significa que Ridge nos da un mejor modelo que Lasso para este

conjunto de datos. Mostraremos una tabla con las métricas de evaluación correspondientes.

	precision	recall	f1-score	support
0.0	0.73	0.67	0.70	91
1.0	0.74	0.79	0.76	108
micro avg	0.73	0.73	0.73	199
macro avg	0.73	0.73	0.73	199
weighted avg	0.73	0.73	0.73	199

Figura 25. Resultado de métricas para Lasso

A continuación se graficara la matriz de confusión del modelo con Lasso:

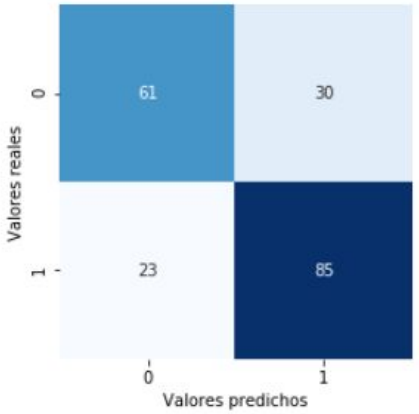


Figura 26. Matriz de confusión para Lasso

Procederemos a realizar una regresión logística tradicional a continuación y mediremos su exactitud con las métricas de sklearn.

```
In [43]: # Regresión Logistica
         from sklearn import linear_model
         from sklearn import model_selection
         # Modelo de regresión logistica
         modeloRL = linear_model.LogisticRegression()
         # Entrenamiento del modelo
         modeloRL.fit(xtrain, ytrain)

Out[43]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='warn',
                   n_jobs=None, penalty='l2', random_state=None, solver='warn',
                   tol=0.0001, verbose=0, warm_start=False)
```

Figura 27. Código para clasificación Logística

De la cual, obtuvimos un porcentaje de exactitud del 62.81%, esto puede ser debido a que no utilizamos regularización para los pesos obtenidos. Para visualizar la clasificación realizada utilizaremos la matriz de confusión:

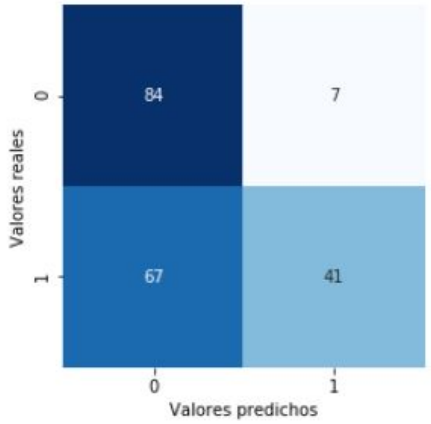


Figura 28. Matriz de confusión para Regresión Logística

Además mostraremos una tabla con las respectivas métricas de evaluación:

	precision	recall	f1-score	support
0.0	0.56	0.92	0.69	91
1.0	0.85	0.38	0.53	108
micro avg	0.63	0.63	0.63	199
macro avg	0.71	0.65	0.61	199
weighted avg	0.72	0.63	0.60	199

Figura 29. Resultado de métricas para Regresión Logística

A continuación, entrenaremos el resultado obtenido de entrenar a nuestros datos con Árboles de Decisiones y con Random Forest
Para árboles de decisiones, utilizaremos 17 particiones, con las cuales obtendremos un porcentaje de exactitud de 63.81%
Graficamos la matriz de confusión del modelo a continuación:

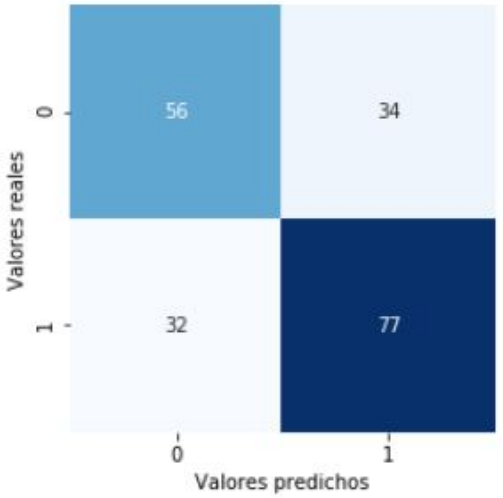


Figura 30. Matriz de confusión para Árbol de decisiones

La tabla de métricas de evaluación es la siguiente:

	precision	recall	f1-score	support
0.0	0.61	0.59	0.60	91
1.0	0.66	0.68	0.67	108
micro avg	0.64	0.64	0.64	199
macro avg	0.64	0.63	0.63	199
weighted avg	0.64	0.64	0.64	199

Figura 31. Resultado de métricas para Árbol de decisiones

Para random forest, importamos BaggingClassifer usando los métodos de ensamble de Sklearn, para ello utilizamos 1000 estimadores para el ensamble, un muestreo máximo de 0.6 muestras de X para el bootstrap y un random_state de 1, lo cual nos sirve para siempre generar el mismo conjunto de datos tomado de manera aleatoria. Con esto obtuvimos un porcentaje de exactitud del 69.34%, a continuación se muestra la matriz de confusión del modelo:

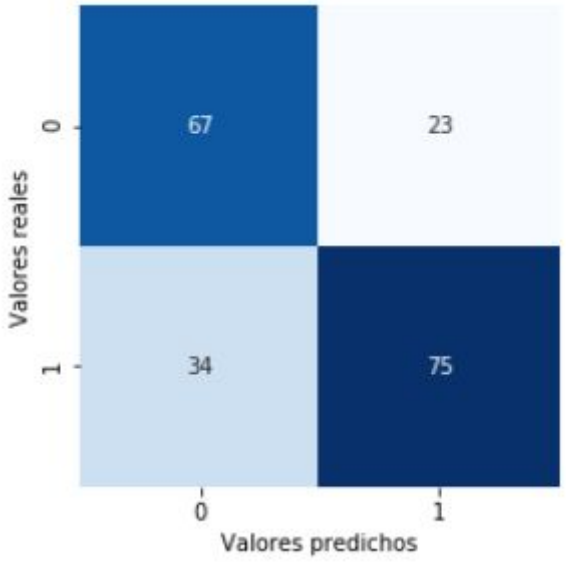


Figura 31. Matriz de confusión para Random Forrest

La tabla con las métricas de evaluación es la siguiente:

	precision	recall	f1-score	support
0.0	0.67	0.66	0.66	91
1.0	0.72	0.72	0.72	108
micro avg	0.69	0.69	0.69	199
macro avg	0.69	0.69	0.69	199
weighted avg	0.69	0.69	0.69	199

Figura 32. Resultado de métricas para Random Forrest.

3.2.3. Resultados por los métodos de Sklearn

A continuación haremos un recuento de los accuracy score de todos los métodos utilizados:

Regresión Lineal de Ridge = 75.37%
Regresión Lineal de Lasso = 73.36%
Regresión Logística = 62.81%
Árboles de decisiones = 63.81%
Random Forest = 69.34%

Podemos ver que el mejor modelo de regresión obtenido es el de regresión lineal de Ridge, esto puede ser debido a que no se transforma ninguno de los pesos a 0, por lo que estos no presentan discontinuidades al momento de hacer la predicción. Podemos ver que aunque los árboles de decisiones causen overfitting, estos tienen un límite al momento de realizar una predicción, de la misma manera sucede con el random forest.

3.3. Método en Python: Método Multiclase

Para la clasificación multiclase se usaron 11 atributos los cuales son apreciados en la Figura 33. Estos atributos son características esenciales para la clasificación de un género musical, además es necesario especificar que los únicos atributos que son vectores son los MFCCs y Vector de Croma cada uno correspondiente a un vector de 13 atributos.

	Atributos
1	Tasa de cruce por cero
2	Energía
3	Entropía de la energía
4	Centroide espectral
5	Propagación espectral
6	Entropía del espectro
7	Flujo espectral
8	Atenuación espectral
9	MFCCs
10	Vector de Croma
11	BPM

Figura 33. Atributos de clasificación multiclase

3.3.1 Definición de los atributos

1. Tasa de cruce por cero

La tasa de cruce por cero (ZCR) de una trama de audio es la tasa de cambios de signo de la señal durante la trama. En otras palabras, es el número de veces que la señal cambia de valor, de positivo a negativo y viceversa, dividido por la longitud del cuadro. La ZCR se define de acuerdo con la siguiente ecuación:

$$Z(i) = \frac{1}{2W_L} \sum_{n=1}^{W_L} | \, sgn[x_i(n)] - sgn[x_i(n-1)] \, |,$$

2. Energía

La energía se define como la suma de los cuadrados de los valores de la señal normalizados al tamaño de la ventana que estamos analizando. Este valor identifica secciones dentro de la señal de audio, que tienen mayor o menor amplitud y viene definido por:

$$E(i) = \sum_{n=1}^{W_L} |x_i(n)|^2.$$

3. Entropía de la energía

La entropía de energía a corto plazo se puede interpretar como una medida de cambios abruptos en el nivel de energía de una señal de audio.

$$H(i) = -\sum_{j=1}^{K} e_j \cdot \log_2 (e_j).$$

$$e_j = \frac{E_{subFrame_j}}{E_{shortFrame_i}},$$

$$E_{shortFrame_i} = \sum_{k=1}^{K} E_{subFrame_k}.$$

4. Centroide espectral

El centroide espectral es el centro de "gravedad" del espectro. Se define como:

$$C_i = \frac{\sum_{k=1}^{Wf_L} kX_i(k)}{\sum_{k=1}^{Wf_L} X_i(k)}.$$

5. Propagación espectral

La propagación espectral es el segundo momento central del espectro. Para calcularlo, se debe tomar la desviación del espectro del centroide espectral, de acuerdo con la siguiente ecuación:

$$S_i = \sqrt{\frac{\sum_{k=1}^{Wf_L} (k - C_i)^2 X_i(k)}{\sum_{k=1}^{Wf_L} X_i(k)}}.$$

6. Entropía del espectro

La entropía espectral se calcula de manera similar a la entropía de la energía, aunque, esta vez, el cálculo se realiza en el dominio de la frecuencia. Se define como:

$$H = -\sum_{f=0}^{L-1} n_f \cdot \log_2 (n_f).$$

7. Flujo espectral

El flujo espectral mide el cambio espectral entre dos cuadros sucesivos y se calcula como la diferencia al cuadrado entre las magnitudes normalizadas de los espectros de las dos ventanas sucesivas a corto plazo:

$$Fl_{(i,i-1)} = \sum_{k=1}^{Wf_L} (EN_i(k) - EN_{i-1}(k))^2,$$

8. Atenuación espectral

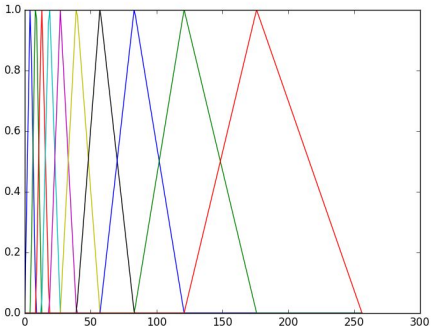
Esta característica se define como la frecuencia por debajo de la cual se concentra un cierto porcentaje (generalmente alrededor del 90%) de la distribución de magnitud del espectro. Por lo tanto, si el m-ésimo coeficiente DFT corresponde a la caída espectral del i-ésimo marco, entonces satisface la siguiente ecuación:

$$\sum_{k=1}^{m} X_i(k) = C \sum_{k=1}^{Wf_L} X_i(k),$$

9. Los coeficientes cepstrales de frecuencia Mel

Los coeficientes de cepstrum de frecuencia Mel (MFCC) han sido muy populares en el campo del procesamiento de voz [5]. Los MFCC son en realidad un tipo de representación cepstral de la señal, donde las bandas de

frecuencia se distribuyen de acuerdo con la escala mel, en lugar del enfoque espaciado linealmente.
En muchas aplicaciones de procesamiento de música se ha vuelto habitual seleccionar los primeros 13 MFCC porque se considera que contienen suficiente información discriminativa en el contexto de varias tareas de clasificación.



10. Vector Croma

El vector de croma es una representación de 12 elementos de la energía espectral. Este es un descriptor ampliamente utilizado, principalmente en aplicaciones relacionadas con la música. El vector de croma se calcula agrupando los coeficientes DFT de un ventana a corto plazo en 12 contenedores. Cada contenedor representa una de las 12 clases de tonos de temperamento igual de música de tipo occidental (espaciado de semitonos). Cada intervalo produce la media de las magnitudes logarítmicas de los respectivos coeficientes DFT:

$$v_k = \sum_{n \in S_k} \frac{X_i(n)}{N_k}, \quad k \in 0, \ldots, 11,$$

11. BPM

Es una unidad de medida del tiempo utilizada en música que indica los pulsos que hay en un minuto. Música más rápida tendrá valores mayores, mientras que música más lenta los tendrá menores.

3.3.2 Obtención de datos

Se usó una función de sklearn para la obtención de los atributos, esta función tiene el nombre de ‘stfeaturesextraction’ . Esta función daba como resultado una matriz de 1200 datos por los 35 atributos (Figura 34), entonces al tener una data extensa pero con datos pertinentes de cada atributo fue necesario realizar la media y la desviación estándar; al realizar esto se obtuvo por cada canción una matriz de 1x35.

Figura 34. Matriz features extraída por cada canción

3.3.3 Dificultades en Clasificación multiclase

Como se mencionó anteriormente, se hizo uso de 11 atributos para poder clasificar adecuadamente 7 géneros musicales (blues,jazz,classical,...,rock). Estos atributos seleccionados dependen seriamente de la calidad de la canción, en otras palabras , dependen de cuánto una canción se identifica con su género y se mostrará porque esto es uno de los problemas más comunes al momento de clasificar géneros musicales.
En primer lugar, tenemos en la Figura. 35 un plano de dispersión donde cada canción está relacionada con su respecto BPM. Las clases se pueden identificar fácilmente teniendo en cuenta el eje x donde cada 100 canciones corresponde a 1 clase.

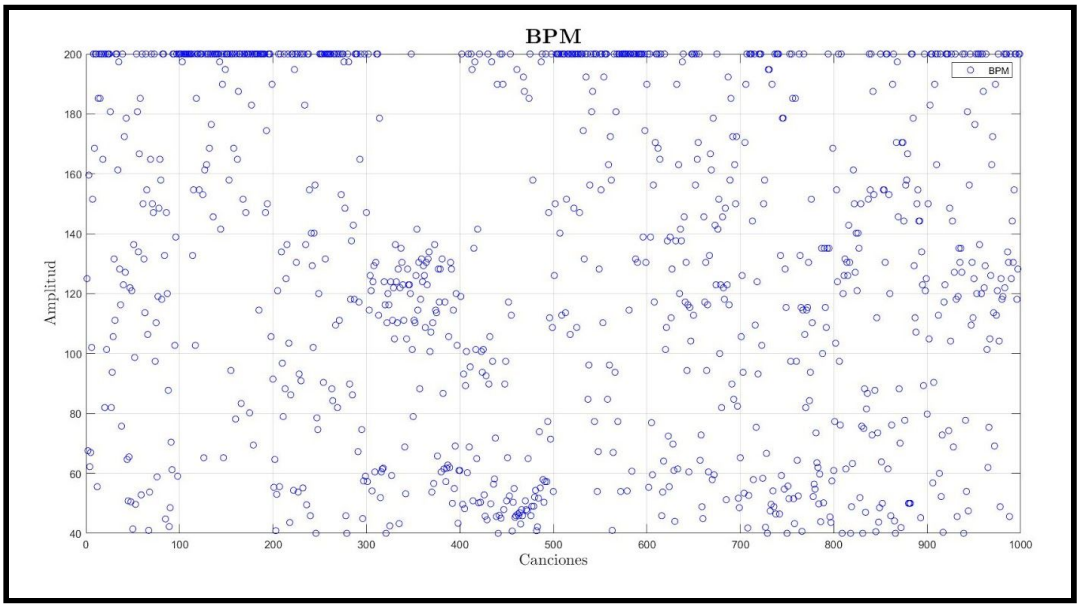


Figura 35. Dispersión de datos de BPM

Como forma ideal se debería tener una dispersión notoria relacionada a cada clase, pero es difícil encontrar cúmulos de datos BPM que pueda interpretarse como una clase, es por ello que en la Figura 36. se muestran en círculo rojo los conjuntos que tienen una característica de High, medium y low respecto a una clase. Esto ayuda a diferenciar claramente 6 géneros musicales, pero los otros 4 géneros es difícil tener clara su clase porque sus respectivos BPMs son muy dispersos y no presentan acumulación de datos como para que se considere cierta particularidad. Empezando desde el punto de particularidad tenemos no sólo 4 géneros fantasmas sino que en los 6 géneros restantes hay similaridad y es notorio que para High tenemos a las clases Classical (100->200) y Jazz(500->600); para Medium, Disco (300->400) y Rock(900->1000); y por último en Low, Hip Hop (400->500) y Pop (700->800).

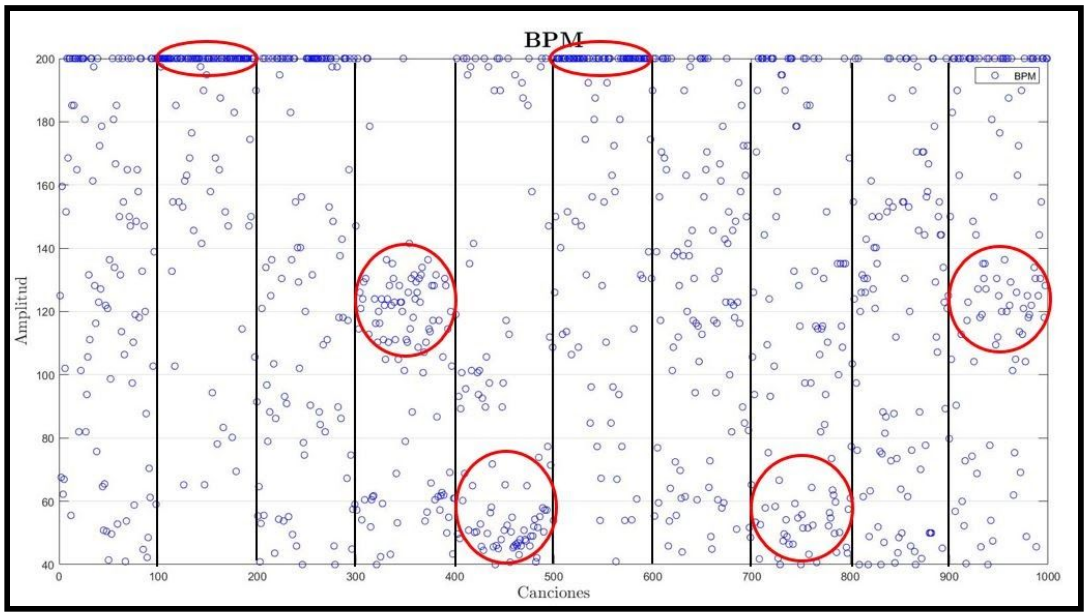


Figura 36. Similaridad en dispersión de datos BPM

Otro serio problema se encuentra en los atributos sacados por Media y Standard Deviation. Se observa en la Figura. 37 en el eje y que desde 0 hacia los negativos son valores comunes y de difícil clasificación porque se encuentran en un mismo rango. También se observa que por encima de una amplitud de 100 los atributos por Media y Standard Deviation tienen un particularidad senoidal.

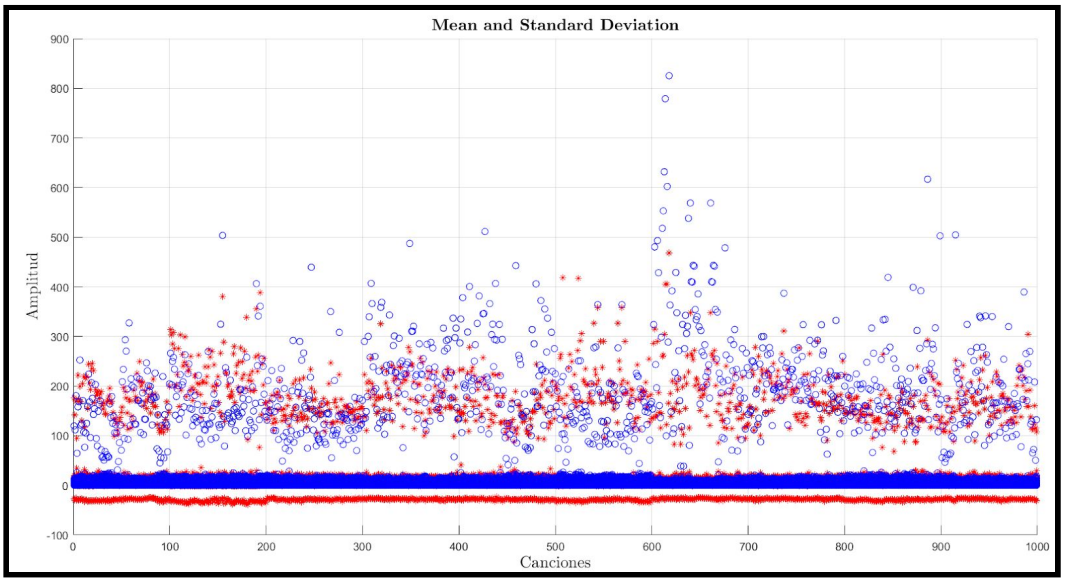


Figura 37. Dispersión de datos de Media y desviación estándar

En la Figura 38 se visualiza claramente la forma senoidal y no dá con claridad una característica primordial respecto a cada clase. Al momento de realizar la clasificación, los picos altos correspondientes a 4 géneros pueden interpretarse de 4 maneras distintas, en lugar de seleccionar una clase en específico. El mismo caso sucede para los picos bajos .

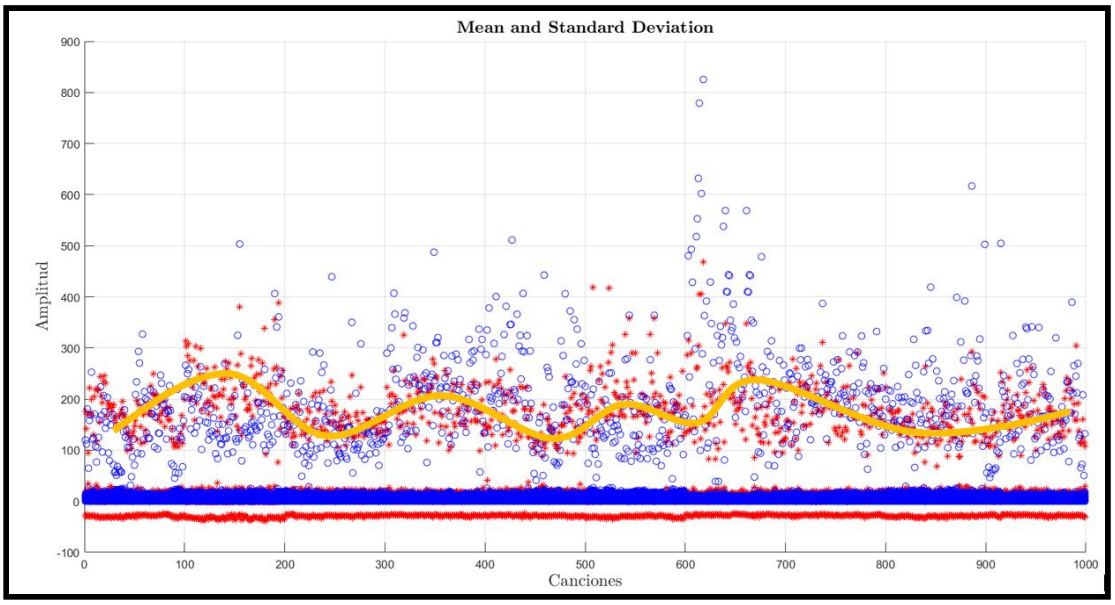


Figura 38. Camino senoidal en dispersión de datos de Media y desviación estándar

Teniendo claro lo explicado anteriormente, se procede a analizar la gráfica de atributos. En la Figura 39 se observa ciertos rangos con cota a 0, estos vendrían a ser los atributos desde 1 hasta 22. Estos atributos a simple vista podrían no considerarse buenos para clasificación pero en la práctica si es de mucha importancia.

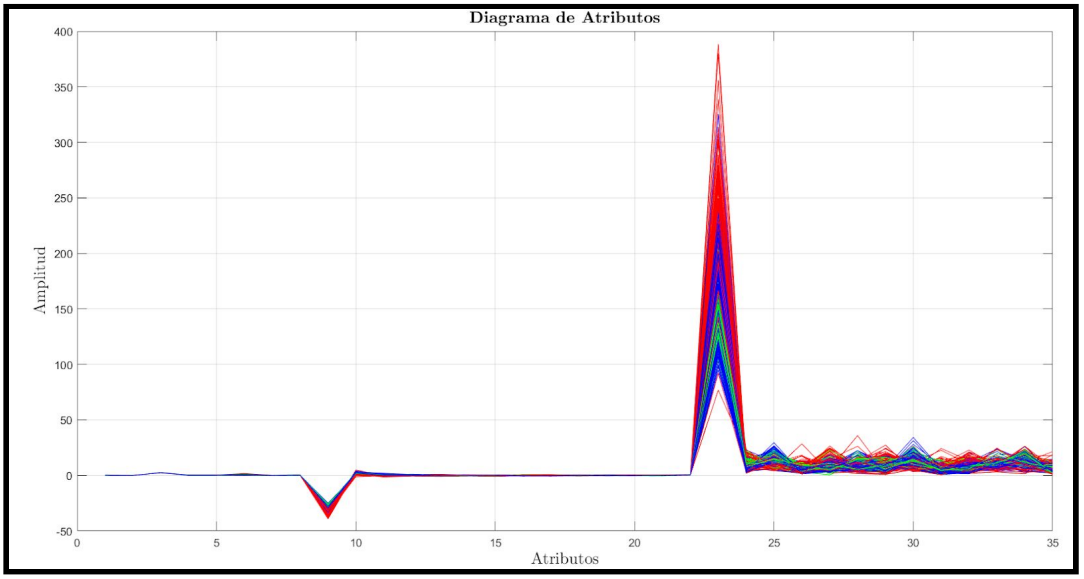


Figura 39. Valores de atributos por cada canción

3.3.4. Resultados de los métodos multiclase

Se decidió aplicar un entrenamiento con distintas formas de atributos incluyendo primero atributos mediante la media, desviación estándar, media + BPM,etc.

MEAN:

Se puede observar en la Figura 40 el resultado de aplicar 3 diferentes tipos de clasificación, así como diversos tipos de métricas. Es de importancia mencionar que el enfoque será en el accuracy y claramente usando Random Forrest se obtiene el mejor resultado para este tipo de atributo.

	MÉTRICAS								
Class	Árbol de decisiones			Random Forrest			K-NN		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
1	0.42	0.46	0.44	0.70	0.75	0.72	0.67	0.50	0.57
2	0.73	0.73	0.73	0.71	0.91	0.80	0.94	0.77	0.85
3	0.40	0.24	0.30	0.62	0.48	0.54	0.47	0.64	0.54
4	0.34	0.48	0.40	0.59	0.55	0.57	0.32	0.39	0.35
5	0.42	0.37	0.39	0.64	0.66	0.65	0.58	0.54	0.56
6	0.43	0.52	0.47	0.63	0.76	0.69	0.58	0.84	0.69
7	0.62	0.77	0.69	0.72	0.97	0.83	0.64	0.93	0.76
8	0.54	0.42	0.47	0.75	0.58	0.65	0.64	0.58	0.61
9	0.53	0.47	0.50	0.65	0.59	0.6	0.56	0.41	0.47
10	0.32	0.32	0.32	0.58	0.48	0.53	0.62	0.26	0.36
Accuracy		0.47			0.66			0.57	

Figura 40. Resultado de métricas para 3 clasificadores

Mediante matrices de confusión de la Figura 41 se pretende dar a conocer la cantidad de datos clasificados correctamente por cada clase. En el caso de Árbol de decisiones la cantidad total de valores clasificados correctamente es 140; para Random Forrest, 199; y para K-NN, 171

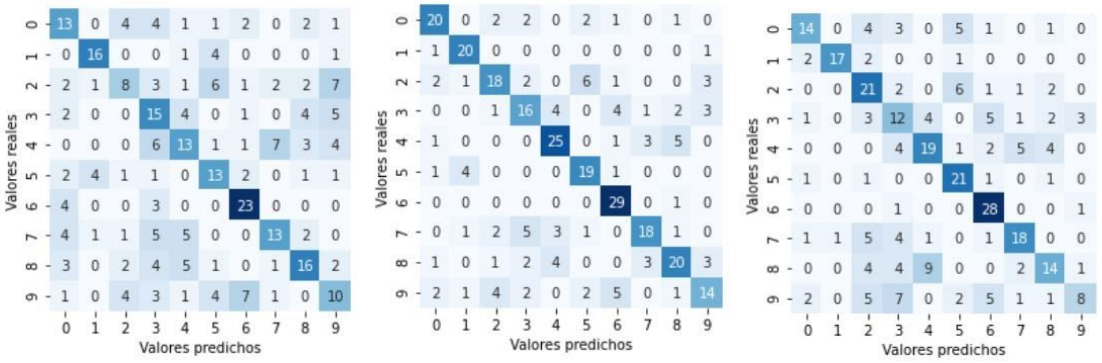


Figura 41. Árbol de decisiones(1), Random Forrest(2), K-NN(3)

MEAN+BPM

Añadiendo un atributo más a los datos de la media ,BPM, se observa que la respuesta en el accuracy no es tan significativa, no se obvia decir que ha aumentado pero el resultado no ha cambiado demasiado respecto a los resultados de MEAN teniendo en cuenta sólo un aumento de 1 a 2 puntos.

MÉTRICAS									
Class	Árbol de decisiones			Random Forrest			K-NN		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
1	0.50	0.54	0.52	0.73	0.68	0.70	0.88	0.50	0.64
2	0.73	0.73	0.73	0.74	0.91	0.82	0.90	0.82	0.86
3	0.47	0.24	0.32	0.68	0.52	0.59	0.46	0.58	0.51
4	0.41	0.48	0.44	0.60	0.58	0.59	0.40	0.55	0.47
5	0.43	0.37	0.40	0.70	0.74	0.72	0.69	0.57	0.62
6	0.39	0.52	0.45	0.61	0.80	0.69	0.56	0.76	0.64
7	0.69	0.90	0.78	0.72	0.93	0.81	0.59	0.90	0.71
8	0.65	0.42	0.51	0.73	0.61	0.67	0.73	0.61	0.67
9	0.62	0.44	0.52	0.63	0.56	0.59	0.55	0.47	0.51
10	0.21	0.32	0.25	0.59	0.52	0.55	0.47	0.26	0.33
Accuracy		0.48			0.67			0.59	

Figura 42. Resultado de métricas para 3 clasificadores

Mediante las matrices de confusión de la Figura 43 se pretende dar a conocer la cantidad de datos clasificados correctamente por cada clase. En el caso de Árbol de decisiones la cantidad total de valores clasificados correctamente es 144; para Random Forrest, 201; y para K-NN, 177

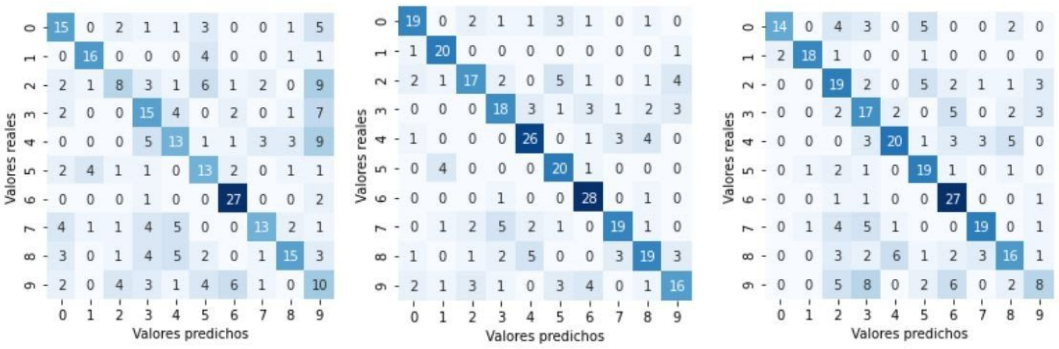


Figura 43. Árbol de decisiones(1), Random Forrest(2), K-NN(3)

STD

Cambiando los atributos dados por desviación estándar se observa en la Figura 44 que el cambio en el accuracy sigue sin ser significativo, simplemente ha mantenido un accuracy similar al Mean+BPM, pero en Árbol de decisiones ha disminuido el valor en 7 puntos. Es evidente que Random Forrest está teniendo un valor mucho mejor respecto a los otros clasificadores.

MÉTRICAS									
Class	Árbol de decisiones			Random Forrest			K-NN		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
1	0.21	0.39	0.27	0.70	0.68	0.69	0.75	0.43	0.55
2	0.68	0.59	0.63	0.82	0.82	0.82	0.82	0.82	0.82
3	0.37	0.33	0.35	0.58	0.64	0.61	0.50	0.67	0.57
4	0.29	0.35	0.32	0.44	0.48	0.46	0.33	0.48	0.39
5	0.42	0.23	0.30	0.71	0.69	0.7	0.77	0.49	0.60
6	0.41	0.52	0.46	0.68	0.76	0.72	0.71	0.60	0.65
7	0.82	0.60	0.69	0.69	0.90	0.78	0.64	0.90	0.75
8	0.46	0.42	0.44	0.85	0.71	0.77	0.53	0.68	0.59
9	0.54	0.41	0.47	0.63	0.65	0.64	0.67	0.59	0.62
10	0.30	0.32	0.31	0.53	0.32	0.40	0.65	0.35	0.46
Accuracy		0.41			0.66			0.59	

Figura 44. Resultado de métricas para 3 clasificadores

Mediante las matrices de confusión de la Figura 45 se pretende dar a conocer la cantidad de datos clasificados correctamente por cada clase. En el caso de Árbol de decisiones la cantidad total de valores clasificados correctamente es 123; para Random Forrest, 198; y para K-NN, 177.

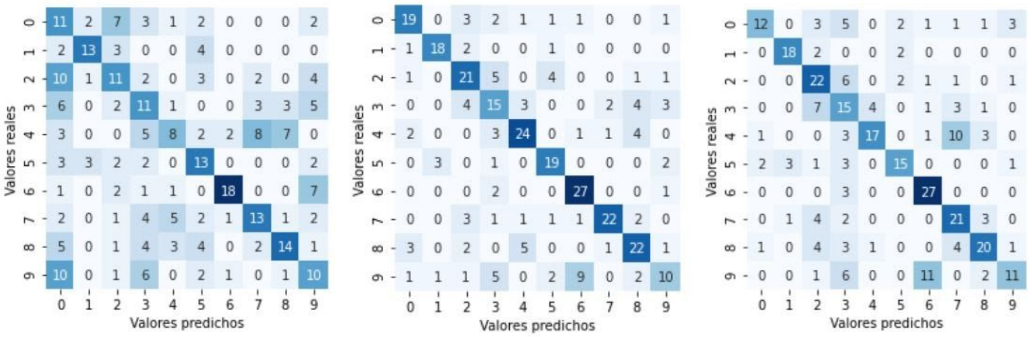


Figura 45. Árbol de decisiones(1), Random Forrest(2), K-NN(3)

STD+BPM

Añadiendo un atributo más ,BPM, a los atributos de la desviación estándar se sigue sin tener un cambio significativo por lo que respecto a las experiencias pasadas se cree conveniente no usar BPM como atributo para clasificar porque no representa un gran cambio y además se ahorra un atributo y se disminuye el costo computacional.

	MÉTRICAS								
Class	Árbol de decisiones			Random Forrest			K-NN		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
1	0.28	0.46	0.35	0.69	0.71	0.70	0.68	0.46	0.55
2	0.75	0.68	0.71	0.86	0.82	0.84	0.79	0.86	0.83
3	0.55	0.33	0.42	0.57	0.61	0.59	0.55	0.67	0.60
4	0.30	0.29	0.30	0.52	0.52	0.52	0.32	0.61	0.42
5	0.39	0.43	0.41	0.75	0.69	0.72	0.68	0.43	0.53
6	0.74	0.48	0.45	0.72	0.84	0.78	0.68	0.52	0.59
7	0.55	0.67	0.7	0.68	0.90	0.77	0.63	0.80	0.71
8	0.52	0.35	0.43	0.75	0.68	0.71	0.53	0.61	0.57
9	0.27	0.41	0.46	0.65	0.65	0.65	0.66	0.56	0.60
10	0.27	0.39	0.32	0.48	0.32	0.38	0.50	0.23	0.31
Accuracy		0.44			0.66			0.57	

Figura 46. Resultado de métricas para 3 clasificadores

Mediante las matrices de confusión de la Figura 47 se pretende dar a conocer la cantidad de datos clasificados correctamente por cada clase. En el caso de Árbol de decisiones la cantidad total de valores clasificados correctamente es 132; para Random Forrest, 198; y para K-NN, 171.

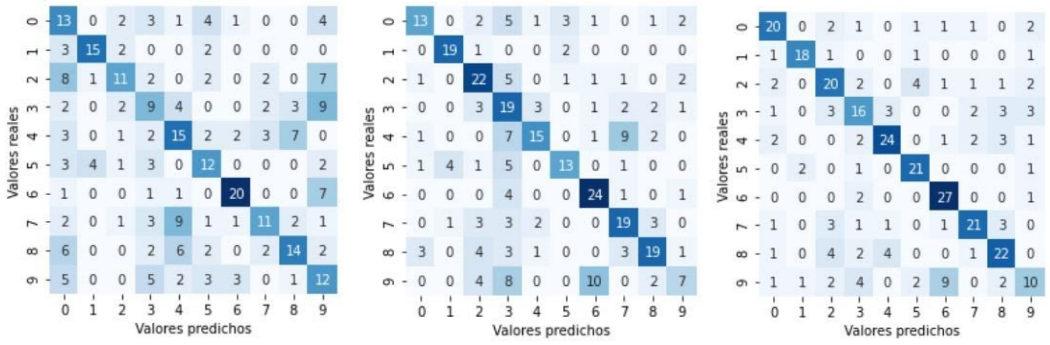


Figura 47. Árbol de decisiones(1), Random Forrest(2), K-NN(3)

STD+MEAN

Ahora se pretende unir los atributos resultado de la media y desviación estándar, esto con el fin de obtener mejores resultados. Es de tener en cuenta que a mayor atributos no significan relativamente mejores resultados, además de ocasionar un costo computacional mayor. Se observa que existe un cambio significativo en los valores de accuracy respecto a los 3 clasificadores, en árbol de decisiones a logrado llegar a 50%; en Random Forrest, superar 70%; y K-NN, superar 60%. Teniendo este resultado es fácil concluir que mezclar ciertos atributos con la media y desviación estándar genera mejores resultados por lo que el siguiente paso sería encontrar los atributos más pertinentes para así reducir dimensionalidad, optimizar código y disminuir costo computacional.

MÉTRICAS									
Class	Árbol de decisiones			Random Forrest			K-NN		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
1	0.48	0.36	0.41	0.88	0.79	0.83	0.71	0.71	0.71
2	0.74	0.77	0.76	0.76	0.86	0.81	0.89	0.73	0.80
3	0.50	0.36	0.42	0.64	0.64	0.64	0.50	0.82	0.62
4	0.26	0.29	0.27	0.63	0.55	0.59	0.32	0.35	0.34
5	0.55	0.49	0.52	0.73	0.69	0.71	0.66	0.54	0.59
6	0.46	0.48	0.47	0.70	0.84	0.76	0.74	0.68	0.71
7	0.57	0.87	0.68	0.76	0.93	0.84	0.72	0.93	0.81
8	0.70	0.52	0.59	0.74	0.65	0.69	0.57	0.65	0.61
9	0.57	0.47	0.52	0.36	0.65	0.64	0.59	0.38	0.46
10	0.33	0.45	0.38	0.68	0.61	0.64	0.78	0.45	0.57
Accuracy		0.5			0.71			0.62	

Figura 48. Resultado de métricas para 3 clasificadores

Mediante las matrices de confusión de la Figura 49 se pretende dar a conocer la cantidad de datos clasificados correctamente por cada clase. En el caso de Árbol de decisiones la cantidad total de valores clasificados correctamente es 150; para Random Forrest, 213; y para K-NN, 186.

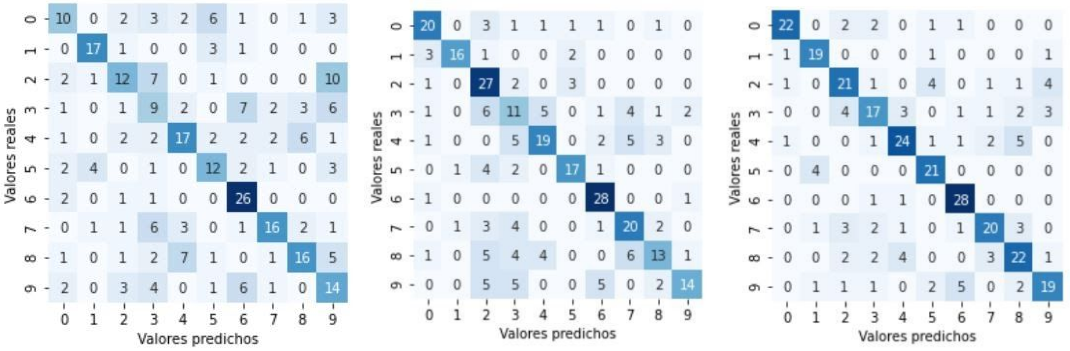


Figura 49. Árbol de decisiones(1), Random Forrest(2), K-NN(3)

Conclusiones

- Se concluye que el mejor clasificador de género musicales es Random Forrest, porque evitó con éxito overfifting con una exactitud de aproximadamente 71%. Además es recomendable usar en un clasificador los atributos relacionados a la media y desviación estándar.

- El clasificador según su carácter de canciones alcanzó una exactitud de 80% aproximadamente

Bibliografía

- Sánchez Hidalgo, A. Multi-label Music Genre Classification. 2019. http://oa.upm.es/54437/1/TESIS_MASTER_ALVARO_SANCHEZ_HIDALGO.pdf
- Beat per Minutes calculation. https://github.com/ederwander/Beat-Track/
- Haytham Fayek. Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. 2016. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html
- Caparrini López, A. Pérez Molina, L. Clasificación de subgéneros de música electrónica. 2017. https://eprints.ucm.es/44672/1/TFG_2017-Clasificacion_subgeneros_musica_electronica.pdf
- Giannakopoulos, T. 2020. Feature Extraction. https://github.com/tyiannak/pyAudioAnalysis/wiki/3.-Feature-Extraction
- Tepepa Cantero, A. Manuel Pérez, H. Nakano Miyatake, M.2018. Algoritmos de aprendizaje supervisado para la clasificación de géneros musicales caracterizados mediante modelos estadísticos. https://www.rcs.cic.ipn.mx/2018_147_5/Algoritmos%20de%20aprendizaje%20supervisado%20para%20la%20clasificacion%20de%20generos%20musicales%20caracterizados.pdf
- La.mathworks.com. 2020. *Feature Extraction*. [online] Available at: <https://la.mathworks.com/discovery/feature-extraction.html>. https://la.mathworks.com/matlabcentral/fileexchange/45831-matlab-audio-analysis-library

- La.mathworks.com. 2020. *Feature Extraction*. [online] Available at: <https://la.mathworks.com/discovery/feature-extraction.html>https://la.mathworks.com/discovery/feature-extraction.html
-
- Giannakopoulos, T., 2020. *Introduction To Audio Analysis. A MATLAB Approach-Academic Press*. 1st ed. Boston: ELSEVIER, pp.1-262.