Árbol de Decisión en Python

1 Introducción

Los árboles de decisión son estructuras de partición recursiva que tienen forma de árbol donde los nodos u hojas representan decisiones y la ramas que salen de estos nodos representan las posibles decisiones y opciones.

Dentro del contexto del aprendizaje automático, los árboles de decisión son modelos de aprendizaje supervisado que dividen recursivamente el espacio de características mediante reglas simples. Este método utiliza una estructura jerárquica de nodos (preguntas) y hojas (decisiones) para clasificar instancias o predecir valores continuos.

2 Metodología

Para realizar el ejercicio de regresión logística, tuve que seguir los siguientes pasos: Visualización General de los datos de Entrada, Análisis de los Datos, Limpieza de los Datos, Mapeo de Atributos, Creación del Árbol de Decisión y Evaluación del Árbol.

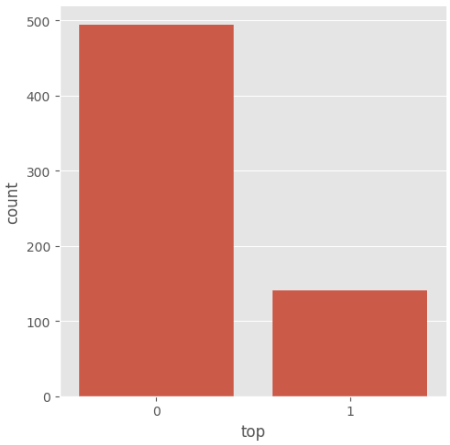
```
#Importamos las librerias
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from IPython.display import Image as PImage
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont

#Creamos un dataframe
artists_billboard = pd.read_csv(r"artists_billboard_fix3.csv")
```

2.1 Visualización General de los datos de Entrada

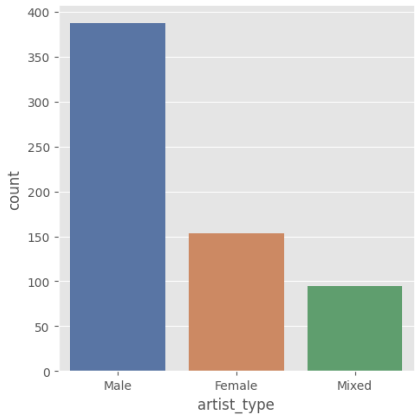
En esta sección vamos a visualizar la distribución de los valores de las columnas categoricas de nuestro *dataset*, y checar que factores influyen en que una canción este en el *top* de las listas.

```
sb.catplot(x='top', data=artists_billboard, kind="count")
```



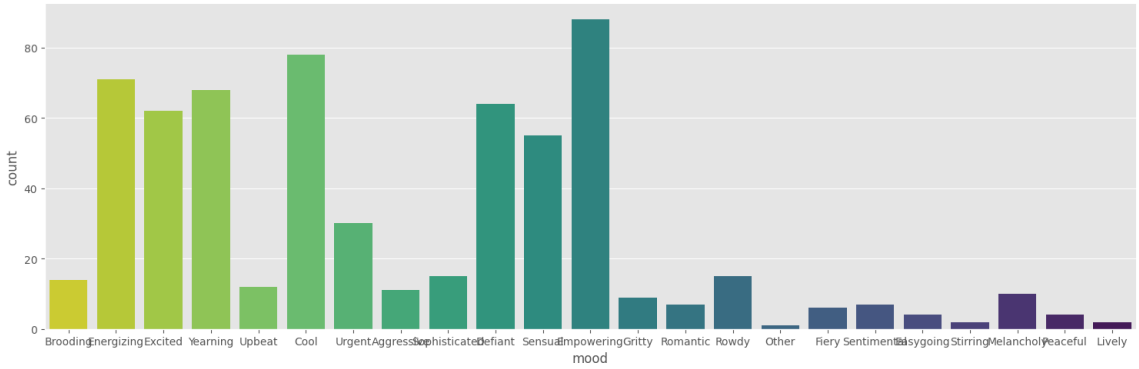
Podemos ver que la mayoría de las canciones en el *dataset* no han estado en el *top*.

```
sb.catplot(x='artist_type',data=artists_billboard, kind="count",
    hue='artist_type', palette='deep')
```



La mayoría de los artistas que estamos tomando en cuenta son hombres, mientras que una minoría son mujeres y canciones de artistsas mixtos.

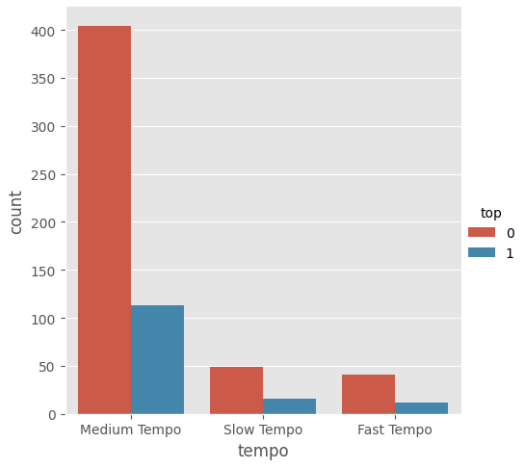
```
sb.catplot(x='mood',data=artists_billboard,kind="count",
    aspect=3, hue='mood', palette='viridis_r')
```



Los sentimientos más comunes en las canciones son aquellos relacionados con emociones positivas.

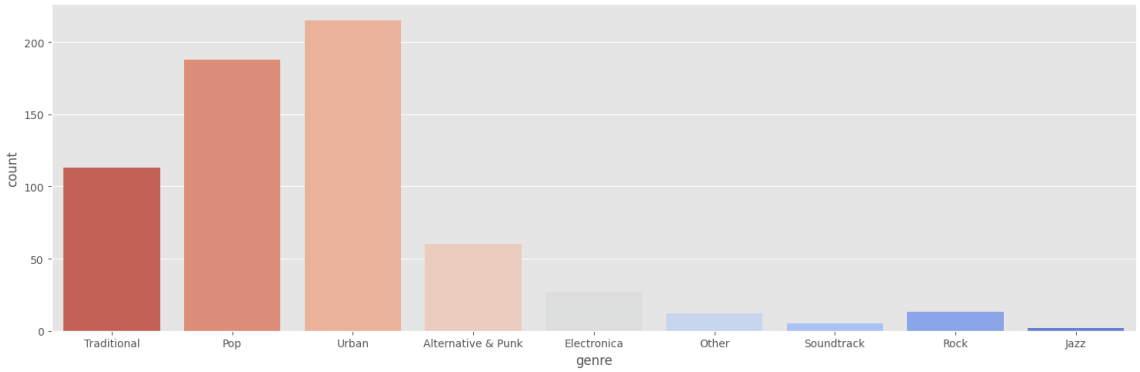
Para analizar el tempo de la canción, vamos a clasificar las canciones con diferentes categorías de tempo en si han estado en el *top* o no.

```
sb.catplot(x='tempo',data=artists_billboard,hue='top',kind="count")
```



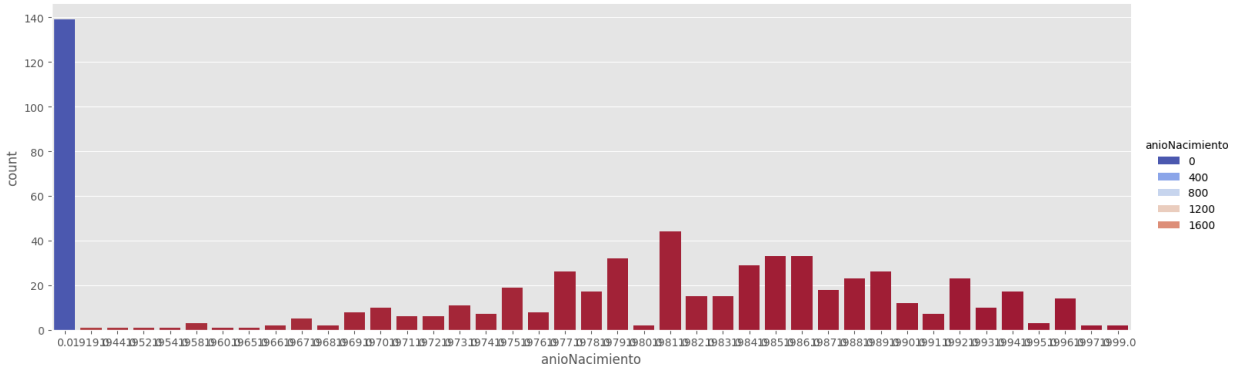
La mayoría de las canciones tienen un tempo normal o medio, mientras que las canciones con tempos rápidos y lentos son menos comunes. Parece que la velocidad del tempo en una canción no influye en si la canción va a estar en el *top*, ya que las tres categorías tienen una proporción similar de canciones en el*top* de las listas.

```
sb.catplot(x='genre',data=artists_billboard,kind="count", aspect=3,
    hue='genre', palette='coolwarm_r')
```



Los géneros más comunes son el Pop, el género urbano y la música tradicional. Los géneros menos comunes son las bandas sonoras, el Jazz y el Rock.

```
sb.catplot(x='anioNacimiento',data=artists_billboard,kind="count",
    aspect=3, hue='anioNacimiento', palette='coolwarm')
```



Aquí podemos notar una anomalía con nuestros datos de la fecha de nacimiento de los artistas, que es que no existe un registro de la fecha de nacimiento para varios artistas, y esos casos tienen un valor de 0. Antes de crear el árbol de decisión se tiene que limpiar estos datos, ya sea poniendo el promedio de todos los datos no nulos o eliminando los registros completamente.

2.2 Análisis de los datos

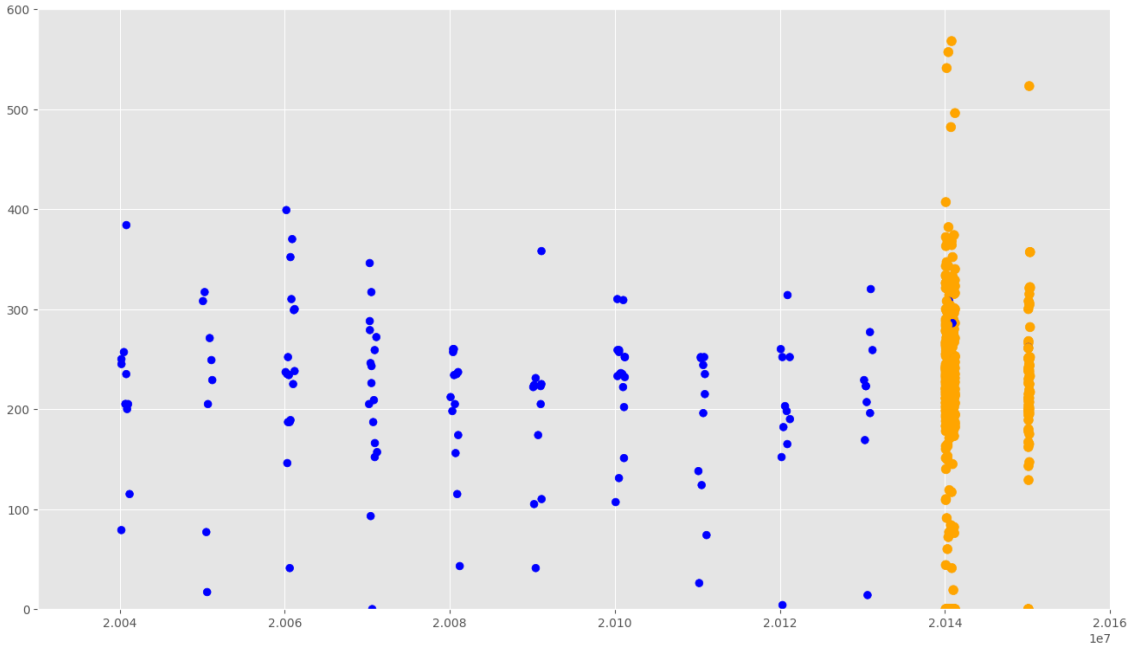
Ya que sabemos la distribución de los datos en las columnas del *dataset*, checamos que columnas tienen un tipo de relación entre sí.

```
f1 = artists_billboard['chart_date'].values
f2 = artists_billboard['durationSeg'].values

colores=['orange','blue']
tamanios=[60,40]

asignar=[]
asignar2=[]
for index, row in artists_billboard.iterrows():
    asignar.append(colores[row['top']])
    asignar2.append(tamanios[row['top']])
```

```
plt.scatter(f1, f2, c=asignar, s=asignar2)
plt.axis([20030101,20160101,0,600])
plt.show()
```



En la gráfica anterior los puntos naranjas representan las canciones *no-top* y los puntos azules representan las canciones *top*. El hecho de que todas las canciones antes de 2014 sean *top* es un problema con la selección de datos en el *dataset*, ya que estos registros fueron agregados después y para balancear la cantidad de canciones *top* y *no-top*. Los datos de 2014 y 2015 muestran una distribución más realista ya que la gran mayoría de las canciones nunca son la canción #1.

Una métrica más útil que el año de nacimiento del artista en el análisis de los factores que hacen que una canción sea #1 es la edad del artista cuando se publico la canción. Antes de calcular este valor se tienen que limpiar los datos del año de nacimiento, para hacer esto se van a asignar edades aleatorias que estén a una desviación aleatoria del promedio de la edad de los artistas. Antes de poder hacer esto, tenemos que calcular la edad promedio de los artistas con fecha de nacimiento no nulo.

```
def edad_fix(anio):
    if anio==0:
        return None
    return anio

artists_billboard['anioNacimiento']=artists_billboard.apply
    (lambda x: edad_fix(x['anioNacimiento']), axis=1);

def calcula_edad(anio,cuando):
    cad = str(cuando)
    momento = cad[:4]
    if anio==0.0:
        return None
    return int(momento) - anio

artists_billboard['edad_en_billboard']=artists_billboard.apply
    (lambda x: calcula_edad(x['anioNacimiento'],x['chart_date']), axis=1);

age_avg = artists_billboard['edad_en_billboard'].mean()
age_std = artists_billboard['edad_en_billboard'].std()
age_null_count = artists_billboard['edad_en_billboard'].isnull().sum()
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std,
    size=age_null_count)
```

```
conValoresNulos = np.isnan(artists_billboard['edad_en_billboard'])

artists_billboard.loc[np.isnan(artists_billboard['edad_en_billboard']),
    'edad_en_billboard'] = age_null_random_list
artists_billboard['edad_en_billboard'] = artists_billboard['edad_en_billboard'].astype(int)
print("Edad Promedio: " + str(age_avg))
print("Desvió Std Edad: " + str(age_std))
print("Intervalo para asignar edad aleatoria: " + str(int(age_avg - age_std))
    + " a " + str(int(age_avg + age_std)))
```

```
Edad Promedio: 30.10282258064516
Desvió Std Edad: 8.40078832861513
Intervalo para asignar edad aleatoria: 21 a 38
```

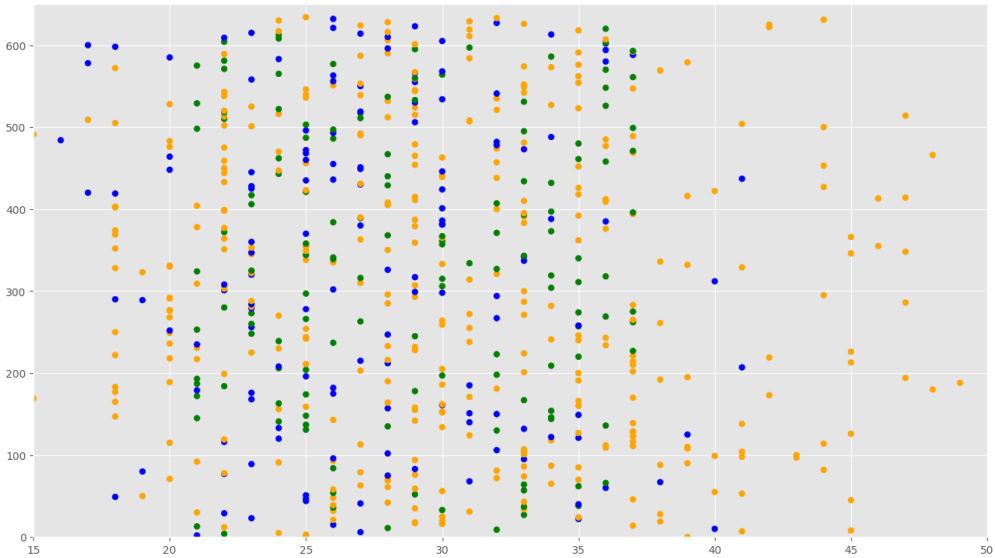
Ahora para visualizar las edades y su relación con la duración de la canción:

```
f1 = artists_billboard['edad_en_billboard'].values
f2 = artists_billboard.index

colores = ['orange','blue','green']

asignar=[]
for index, row in artists_billboard.iterrows():
    if (conValoresNulos[index]):
        asignar.append(colores[2]) # verde
    else:
        asignar.append(colores[row['top']])

plt.scatter(f1, f2, c=asignar, s=30)
plt.axis([15,50,0,650])
plt.show()
```



2.3 Mapeo de Atributos

Se realiza un mapeo de los atributos de entrada para transformarlos en categorías que se puedan utilizar en el árbol de decisión

```
# Mood Mapping
artists_billboard['moodEncoded'] = artists_billboard['mood'].map( {'Energizing': 6,
                                    'Empowering': 6,
```

```
                                        'Cool': 5,
                                        'Yearning': 4, # anhelo, deseo, ansia
                                        'Excited': 5, #emocionado
                                        'Defiant': 3,
                                        'Sensual': 2,
                                        'Gritty': 3, #coraje
                                        'Sophisticated': 4,
                                        'Aggressive': 4, # provocativo
                                        'Fiery': 4, #caracter fuerte
                                        'Urgent': 3,
                                        'Rowdy': 4, #ruidoso alboroto
                                        'Sentimental': 4,
                                        'Easygoing': 1, # sencillo
                                        'Melancholy': 4,
                                        'Romantic': 2,
                                        'Peaceful': 1,
                                        'Brooding': 4, # melancolico
                                        'Upbeat': 5, #optimista alegre
                                        'Stirring': 5, #emocionante
                                        'Lively': 5, #animado
                                        'Other': 0,'':0} ).astype(int)
# Tempo Mapping
artists_billboard['tempoEncoded'] = artists_billboard['tempo'].map( {'Fast Tempo': 0,
                                        'Medium Tempo': 2,
                                        'Slow Tempo': 1, '': 0} ).astype(int)
# Genre Mapping
artists_billboard['genreEncoded'] = artists_billboard['genre'].map( {'Urban': 4,
                                          'Pop': 3,
                                          'Traditional': 2,
                                          'Alternative & Punk': 1,
                                         'Electronica': 1,
                                          'Rock': 1,
                                          'Soundtrack': 0,
                                          'Jazz': 0,
                                          'Other':0,'':0}
                                        ).astype(int)
# artist_type Mapping
artists_billboard['artist_typeEncoded'] = artists_billboard['artist_type'].map( {'Female': 2,
                                          'Male': 3,
                                          'Mixed': 1,
                                          '': 0} ).astype(int)


# Mapping edad en la que llegaron al billboard
artists_billboard.loc[ artists_billboard['edad_en_billboard'] <= 21, 'edadEncoded']
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 21) &
            (artists_billboard['edad_en_billboard'] <= 26), 'edadEncoded'] = 1
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 26) &
            (artists_billboard['edad_en_billboard'] <= 30), 'edadEncoded'] = 2
artists_billboard.loc[(artists_billboard['edad_en_billboard'] > 30) &
            (artists_billboard['edad_en_billboard'] <= 40), 'edadEncoded'] = 3
artists_billboard.loc[ artists_billboard['edad_en_billboard'] > 40, 'edadEncoded'] = 4

# Mapping Song Duration
artists_billboard.loc[ artists_billboard['durationSeg'] <= 150, 'durationEncoded']
artists_billboard.loc[(artists_billboard['durationSeg'] > 150) &
            (artists_billboard['durationSeg'] <= 180), 'durationEncoded'] = 1
artists_billboard.loc[(artists_billboard['durationSeg'] > 180) &
```

```
            (artists_billboard['durationSeg'] <= 210), 'durationEncoded'] = 2
artists_billboard.loc[(artists_billboard['durationSeg'] > 210) &
            (artists_billboard['durationSeg'] <= 240), 'durationEncoded'] = 3
artists_billboard.loc[(artists_billboard['durationSeg'] > 240) &
            (artists_billboard['durationSeg'] <= 270), 'durationEncoded'] = 4
artists_billboard.loc[(artists_billboard['durationSeg'] > 270) &
            (artists_billboard['durationSeg'] <= 300), 'durationEncoded'] = 5
artists_billboard.loc[ artists_billboard['durationSeg'] > 300, 'durationEncoded'] = 6

drop_elements = ['id','title','artist','mood','tempo','genre','artist_type',
            'chart_date','anioNacimiento','durationSeg','edad_en_billboard']
artists_encoded = artists_billboard.drop(drop_elements, axis = 1)
```

2.4 Creación del Árbol de Decisión

En esta parte se crea el árbol de decisión, para procurar su máxima precisión se tiene que analizar el nivel de la profundidad óptimo. Creamos varios prototipos del árbol con diferentes profundidades y calculamos su precisión:

```
cv = KFold(n_splits=10)
accuracies = list()
max_attributes = len(list(artists_encoded))
depth_range = range(1, max_attributes + 1)

for depth in depth_range:
    fold_accuracy = []
    tree_model = tree.DecisionTreeClassifier(criterion='entropy',
                min_samples_split=20, min_samples_leaf=5, max_depth = depth,
                class_weight={1:3.5})
    for train_fold, valid_fold in cv.split(artists_encoded):
        f_train = artists_encoded.loc[train_fold]
        f_valid = artists_encoded.loc[valid_fold]

        model = tree_model.fit(X = f_train.drop(['top'], axis=1),
                               y = f_train["top"])
        valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
                                y = f_valid["top"])
        fold_accuracy.append(valid_acc)

    avg = sum(fold_accuracy)/len(fold_accuracy)
    accuracies.append(avg)

df = pd.DataFrame({"Max Depth": depth_range, "Average Accuracy": accuracies})
df = df[["Max Depth", "Average Accuracy"]]
print(df.to_string(index=False))
```

```
Max Depth  Average Accuracy
    1          0.556101
    2          0.556126
    3          0.564038
    4          0.644122
    5          0.614335
    6          0.631498
    7          0.617336
```

No siempre tener más niveles en un árbol de decisión es mejor, como se puede ver en este caso: la profundidad óptima es 4 niveles con una precisión del 64.41%.

Se crea el árbol de decisión con 4 niveles de profundidad y usamos la aplicación de GraphViz para exportar un diagrama del árbol de decisión y convertirlo en una imagen.

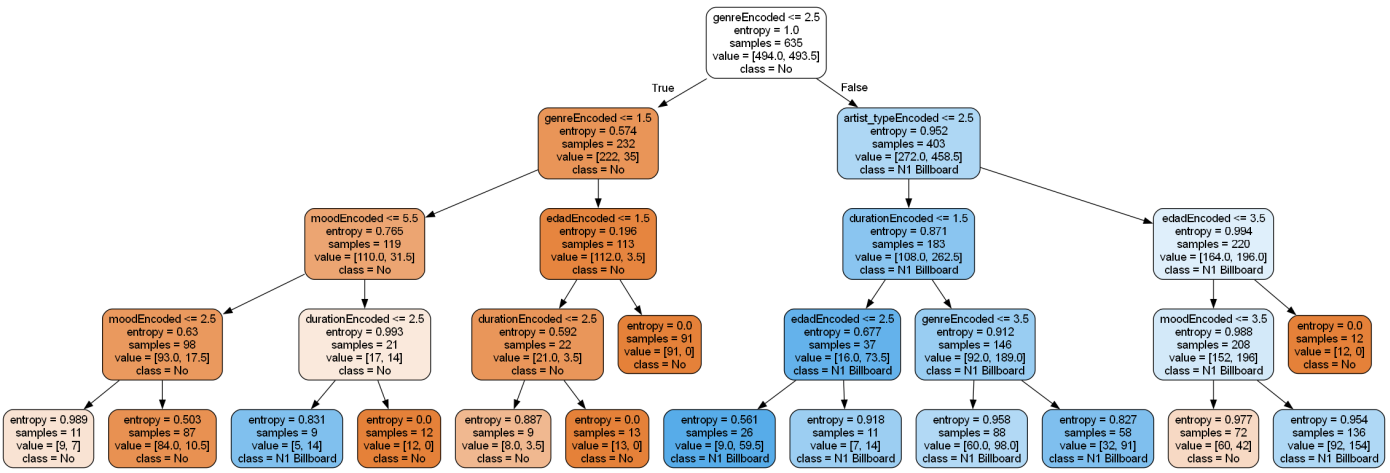
```
y_train = artists_encoded['top']
x_train = artists_encoded.drop(['top'], axis=1).values

decision_tree = tree.DecisionTreeClassifier(criterion='entropy',
                min_samples_split=20, min_samples_leaf=5, max_depth = 4,
                class_weight={1:3.5})
decision_tree.fit(x_train, y_train)

with open(r"tree1.dot", 'w') as f:
     f = tree.export_graphviz(decision_tree, out_file=f, max_depth = 7, impurity = True,
        feature_names = list(artists_encoded.drop(['top'], axis=1)),
        class_names = ['No', 'N1 Billboard'],
        rounded = True,
        filled= True )

import os
os.environ["PATH"] += os.pathsep + r'D:\Graphviz-12.2.1-win64\bin'

check_call(['dot','-Tpng',r'tree1.dot','-o',r'tree1.png'])
PImage("tree1.png")
```




```
acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
print(acc_decision_tree)
```

```
64.88
```

El modelo está en lo correcto en un 64.88

3 Resultados

Para verificar la precisión del modelo, vamos a hacer predicciones de *rankings* de canciones en el *dataset*.

```
#predecir artista CAMILA CABELLO featuring YOUNG THUG
# con su canción Havana llego a numero 1 Billboard US en 2017

x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded', 'genreEncoded',
    'artist_typeEncoded','edadEncoded','durationEncoded'))
x_test.loc[0] = (1,5,2,4,1,0,3)
y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
print("Prediccion: " + str(y_pred))
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred][0]* 100, 2))+"%")
```

```
Prediccion: [1]
Probabilidad de Acierto: 73.98%
```

```
#predecir artista Imagine Dragons
# con su canción Believer llego al puesto 42 Billboard US en 2017

x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded', 'genreEncoded',
    'artist_typeEncoded','edadEncoded','durationEncoded'))
x_test.loc[0] = (0,4,2,1,3,2,3)
y_pred = decision_tree.predict(x_test.drop(['top'], axis = 1))
print("Prediccion: " + str(y_pred))
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
print("Probabilidad de Acierto: " + str(round(y_proba[0][y_pred][0]* 100, 2))+"%")
```

```
Prediccion: [0]
Probabilidad de Acierto: 88.89%
```

4 Conclusión

Los modelos de árboles de decisión son herramientas de predicción usadas en el aprendizaje automático que están basadas en la figura lógica del mismo nombre. Cada nodo representa un evento o decisión con múltiples ramas que puede seguir dependiendo de los datos que se pasan al modelo.

En este caso, se creó un modelo de árbol de decisión que busca saber si una canción estuvo en la posición 1 de las tablas de Billboard dado varias variables que son parte del *dataset*, el modelo regresa una probabilidad de que la canción haya estado en esta posición. Se checo el puntaje de varios modelos con distintas profundidades para procurar que el modelo fuera lo más acertado posible. El modelo final tiene una precisión del 64.88%, lo que significa que por cada predicción incorrecta que realiza, hace otras dos predicciones que son correctas.