Arbol de Decisión

Valeria Ybarra López

29, Marzo 2025

1 Introducción

Los árboles de decisión son representaciones gráficas y algoritmos de aprendizaje supervisado utilizados en machine learning para tareas de clasificación y regresión.

Un árbol de decisión se estructura a partir de un nodo raíz que se divide en ramas según condiciones verdaderas o falsas, hasta llegar a las hojas, que representan soluciones finales. Su propósito es tomar decisiones óptimas desde una perspectiva probabilística, evaluando subdivisiones en los datos para construir el árbol más eficiente.

2 Metodología

Para la realización de esta activdad, se siguieron las instrucciones proporcionadas en la página 51 del libro "Aprenda Machine Learning".

2.1 Creación de carpeta Arból de Decisión

Se creó una carpeta con el nombre de "Arbol de Decisión" en donde se guardó el archivo .csv (que contiene los datos de entrada) proporcionado por el libro para poder realizar el código en python, en esa misma carpeta se creó un archivo .py para realizar la actividad.

2.2 Archivo csv

El archivo csv proporciona datos de cantantes y bandas que lograron un puesto número uno en las listas de Billboard Hot 100, con los cuales intentaremos predecir el próximo al número 1 del Billboard.

2.3 Código

Primero, importaremos las bibliotecas necesarias:

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
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
```

Análisis Exploratorio Inicial

Comenzamos leyendo nuestros datos de entrada, para después ver cuantas columnas y registros se tienen.

```
artists_billboard =
pd.read_csv(r"artists_billboard_fix3.csv")
print(artists_billboard.shape)
```

Veamos los primeros 5 registros:

```
pd.set_option('display.max_columns', None)
print(artists_billboard.head())
```

Al usar .head(), podemos ver que en la tabla tenemos: Titulo de la canción, artista, "mood", tempo, género, Tipo de artista, fecha en que apareció en el billboard, la columna TOP será nuestra etiqueta, en la que aparece 1 si llegó al número uno de Billboard ó 0 si no lo alcanzó y por último, el año de nacimiento del artista.

Seguimos realizando visualizaciones para comprender mejor los datos; usaremos groupby para ver cuántos alcanzaron el número uno y cuantos no:

```
print(artist_billboard.groupby('top').size())
```

Visualicemos los Atributos de entrada:

```
sb.catplot(x='top',
data=artists_billboard,kind='count')
sb.catplot(x='artist_type', data=artists_billboard,
kind='count')
sb.catplot(x='top',
data=artists_billboard,hue='artist_type',kind="count")
sb.catplot(x='mood',
data=artists_billboard,kind="count", aspect=3)
sb.catplot(x='tempo',
data=artists_billboard,hue='top',kind="count")
sb.catplot(x='genre',
data=artists_billboard,kind="count", aspect=3)
sb.catplot(x='mood',
data=artists_billboard,hue='top',
kind="count", aspect=3)
sb.catplot(x='anioNacimiento',
```

```
data=artists_billboard,kind="count", aspect=3)
plt.show()
```

Balanceo de Datos

No hay equilibrio en la cantidad de etiquetas top y no-top de las canciones, debido a que en el transcurso de un año, unas 5 o 6 canciones logran el primer puesto.

Buscamos si hay alguna relación entre Año y duración de la canción:

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

f1 = artists_billboard['anioNacimiento'].values
f2 = artists_billboard['durationSeg'].values

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

plt.scatter(f1, f2, c=asignar, s=30)
plt.axis([1960,2005,0,600])
plt.show()
```

Visualicemos los top y no-top de acuerdo a sus fechas:

```
 f1 = artists_billboard['chart_date'].values
f2 = artists_billboard
['durationSeg'].values tamanios = [60, 40]
colores=['orange','blue']
tamanios = [tamanios[row['top']]
for _, row in artists_billboard.iterrows()]
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()
```

Preparamos los datos

Arreglemos el problema de los años de nacimiento que están en cero:

```
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)
```

Luego calculamos las edades en una nueva columna “edad_en_billboard” restando el año de aparición (los primeros 4 caracteres de chart_date) al año de nacimiento:

```
1
2  def calcula_edad(anio,cuando):
3      cad = str(cuando)
4      momento = cad[:4]
5      if anio==0.0:
6          return None
7      return int(momento) - anio
8
9  artists_billboard['edad_en_billboard']=
10 artists_billboard.apply(lambda x: calcula_edad(x['anioNacimiento'],
11 x['chart_date']), axis=1)
12 artists_billboard.head()
```

Finalmente asigaremos edades aleatorias a los registros faltantes, para esto obtenemos el promedio de edad de nuestro conjunto (avg) y su desvío estándar (std) y pedimos valores random a la función que van desde avg - std hasta avg + std:

```
1  age_avg = artists_billboard
2  ['edad_en_billboard'].mean()
3  age_std = artists_billboard['edad_en_billboard'].std()
4  age_null_count = artists_billboard['edad_en_billboard'].isnull().
       sum()
5  age_null_random_list = np.random.randint
6  (age_avg - age_std, age_avg + age_std,
7  size=age_null_count)
8
9  conValoresNulos = np.isnan
10 (artists_billboard['edad_en_billboard'])
11
12 artists_billboard.loc
13 [np.isnan(artists_billboard['edad_en_billboard']),
14 'edad_en_billboard'] = age_null_random_list
15 artists_billboard['edad_en_billboard'] =
16 artists_billboard['edad_en_billboard'].astype(int)
17
18 print("Edad␣Promedio:␣" + str(age_avg))
19 print("Desvio␣Std␣Edad:␣" +str(age_std))
20 print("Intervalo␣para␣asignar␣edad␣aleatoria:
21 " + str(int(age_avg - age_std)) +
22 "␣a␣" + str(int(age_avg + age_std)))
```

Ahora visualizemos los valores agregados (en color verde) en el siguiente gráfico:

```
1  f1 = artists_billboard
2  ['edad_en_billboard'].values
3  f2 = artists_billboard.index
4  colores = ['orange','blue','green']
5  asignar=[]
6  for index, row in artists_billboard.iterrows():
7      if (conValoresNulos[index]):
8          asignar.append(colores[2]) # verde
9      else:
10      asignar.append(colores[row['top']])
11 plt.scatter(f1, f2, c=asignar, s=30)
```

```
plt.axis([15,50,0,650])
plt.show()
```

Mapeo de Datos

Transformamos varios de los datos de entrada en valores categóricos. Serapamos las edades en: menor de 21 años, entre 21 y 26, etc. Hacemos lo msimo para la duración de canciones. El ´´mood" agrupamos los que son similares. El tempo como: 0-Rapido, 1-Lento, 2-Medio.

```
# 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)
```

Una vez corrido el código con exito, obtenemos un nuevo conjunto de datos llamado artist_encoded en el cual se tienen atributos deinitivos para crear nuestro arból. Para esto quitamos las columnas que no necesitamos con ´´drop":

```
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)
```

Como quedan los top en relación a los datos mapeados

Visualizamos por medio de tablas, la repartición de los top 1 en los diversos atributos mapeados. Sobre la columna sum, estan los top:

```
print(artists_encoded[['moodEncoded', 'top']].groupby(['moodEncoded
    '], as_index=False).agg(['mean', 'count', 'sum']))
print(artists_encoded[['artist_typeEncoded', 'top']].groupby(['
    artist_typeEncoded'], as_index=False).agg(['mean', 'count', '
    sum']))
print(artists_encoded[['genreEncoded', 'top']].groupby(['
    genreEncoded'], as_index=False).agg(['mean', 'count', 'sum']))
print(artists_encoded[['tempoEncoded', 'top']].groupby(['
    tempoEncoded'], as_index=False).agg(['mean', 'count', 'sum']))
print(artists_encoded[['durationEncoded', 'top']].groupby(['
    durationEncoded'], as_index=False).agg(['mean', 'count', 'sum'
    ]))
print(artists_encoded[['edadEncoded', 'top']].groupby(['edadEncoded
    '], as_index=False).agg(['mean', 'count', 'sum']))
```

Buscamos la profundidad para el arból de decisión

Antes de crear el árbol, vamos a buscar cuántos niveles de profundida le asignaremos, pero esto usaremos la función KFold la cual ayuda a crear varios subgrupos con los datos de entrada para validar y valorar los árboles con diversos niveles de profundidad.

Creamos el árbol

Para crear el árbol de clasificación utilizamos la librería de sklearn tree.DecisionTreeClasifier, y lo configuramos con los parámetros:

- `criterion=entropy`
- `min_samples_split=20`: cantidad mínima de muestras que debe tener el nodo para subdividir.
- `min_samples_leaf=5`: cantidad mínima que puede tener la hoja final. Si tiene menos, no se forma la hoja y sube un nivel, su antecesor.
- `class_weight={1:3.5}`: compensamos los desbalances. En este caso, tenemos menos etiquetas de tipo top=1, por lo que asignamos 3.5 de peso a la etiqueta 1. El valor sale de dividir 494 (cantidad de top=0) entre 141 (cantidad de top=1).

```
    cv = KFold(n_splits=10) # Numero deseado de "folds" que haremos
accuracies = list()
max_attributes = len(list(artists_encoded))
depth_range = range(1, max_attributes + 1)

# Testearemos la profundidad de 1 a cantidad de atributos +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]
```

```
17
18          model = tree_model.fit(X = f_train.drop(['top'], axis=1),
19                                 y = f_train["top"])
20          valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
21                                  y = f_valid["top"]) # calculamos la
                                       precision con el segmento de
                                      validacion
22          fold_accuracy.append(valid_acc)
23
24      avg = sum(fold_accuracy)/len(fold_accuracy)
25      accuracies.append(avg)
26
27  # Mostramos los resultados obtenidos
28  df = pd.DataFrame({"Max␣Depth": depth_range, "Average␣Accuracy":
        accuracies})
29  df = df[["Max␣Depth", "Average␣Accuracy"]]
30  print(df.to_string(index=False))
```

Visualización del árbol de decisión

Asignamos los datos de entrada a los parámetros que configuramos anteriormente con 4 niveles de profundidad. Usamos la función de export_graphviz para crear un archivo de extensión .dot que luego convertiremos en un gráfico png para visualizar el árbol.

```
1   # Crear arrays de entrenamiento y las etiquetas que indican si
        lleg  a top o no
2   y_train = artists_encoded['top']
3   x_train = artists_encoded.drop(['top'], axis=1).values
4
5   # Crear Arbol de decision con profundidad = 4
6   decision_tree = tree.DecisionTreeClassifier(criterion='entropy',
7                                               min_samples_split=20,
8                                               min_samples_leaf=5,
9                                               max_depth = 4,
10                                              class_weight={1:3.5})
11  decision_tree.fit(x_train, y_train)
12
13  # exportar el modelo a archivo .dot
14  with open(r"tree1.dot", 'w') as f:
15       f = tree.export_graphviz(decision_tree,
16                                out_file=f,
17                                max_depth = 7,
18                                impurity = True,
19                                feature_names = list(artists_encoded.
                                      drop(['top'], axis=1)),
20                                class_names = ['No', 'N1␣Billboard'],
21                                rounded = True,
22                                filled= True )
23
24  # Convertir el archivo .dot a png para poder visualizarlo
25  check_call(['dot','-Tpng',r'tree1.dot','-o',r'tree1.png'])
26  PImage("tree1.png")
```

Análisis

Una vez creado el árbol,vemos que comienza con un nodo raíz que clasifica por género: los géneros 0, 1 y 2 (menor a 2.5) van a la izquierda (True), mientras

que los géneros 3 y 4 (Pop y Urban, con mayor número de usuarios top Billboard) van a la derecha (False). En el segundo nivel, las muestras se dividen en 232 y 403 respectivamente. A medida que el árbol desciende, los valores de entropía se aproxíman más a 1 cuando hay más muestras top=1 (azul) y se acercan a 0 cuando hay mayoría de muestras top=0 (naranja). En los siguientes niveles, se encuentran las divisiones por tipo de artista, edad, duración y mood. Algunas hojas naranjas se terminan antes del último nivel debido a que alcanzan entropía cero o porque tienen menos de 20 muestras (mínimo necesario para dividir).

Veamos cuál fue la precisión alcanzada por nuestro árbol:

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

Predicción de Canciones al Billboard 100

Testeamos el árbol con 2 artistas que entraron al billboard 100 en 2017: Camila Cabello con la canción Havana (top 1) e Imagine Dragons con la canción Believer (top 42).

```
#predecir artista CAMILA CABELLO featuring YOUNG THUG
# con su canci n 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))+"%")
```

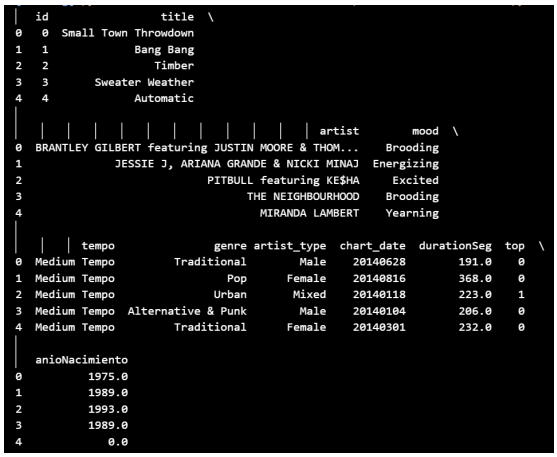
```
#predecir artista Imagine Dragons
# con su cancion Believer llego al puesto 42 Billboard US en 2017
x_test = pd.DataFrame(columns=('top','moodEncoded', 'tempoEncoded',
     'genreEncoded','artist_typeEncoded','edadEncoded','
    durationEncoded'))
x_test.loc[0] = (0,4,2,1,3,2,3)
y_pred = decision_tree.predict(x_test.drop(['top'], 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))+"%")
```

3 Resultados

Análisis Exploratorio Inicial

```
(635, 11)
```

Figure 1: Resultado de .shape(), nos devuelve (635,11) es decir que tenemos 11 columnas(features) y 635 filas de datos



	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	anioNacimiento
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	1975.0
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	1989.0
2	2	Timber	PITBULL featuring KE$HA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1	1993.0
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0	1989.0
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0	0.0

Figure 2: Resultado de .head(), los primeros 5 registros.

```
top
0    494
1    141
dtype: int64
```

Figure 3: Resultado de groupby('top') para ver cuantos alcanzarón el top 1 y cuantos no; tenemos 494 canciones que no lograron el top 1 y 141 que si alcanzaron el top 1.(una cantidad desbalanceada.)

Atributos de entrada

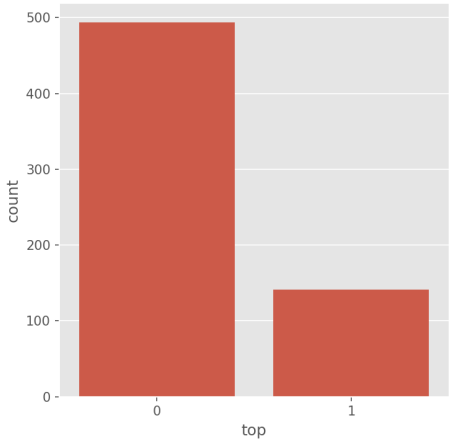


Figure 4: Las etiquetas indican 0-No llego al top 1 y 1-Llego al top 1.

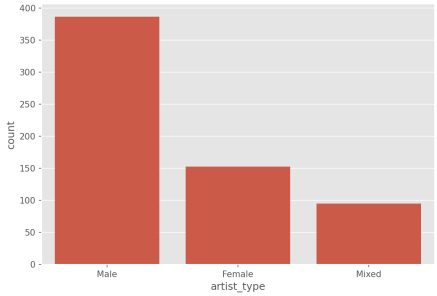


Figure 5: Vemos que hay más artistas masculinos que femeninos, y unos 100 registros de canciones mixtas.

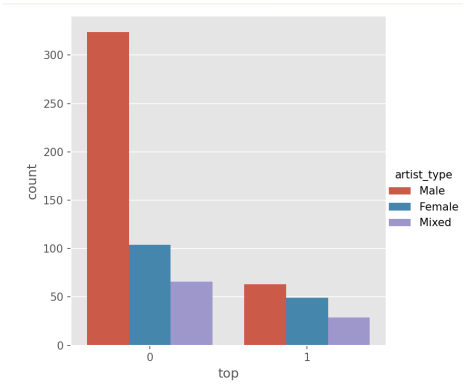


Figure 6: Vemos cuantos artistas femeninos, masculinos, o de ambos, han alcanzado el top 1.

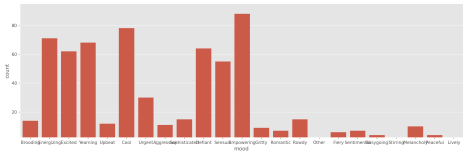


Figure 7: Aquí vemos el "mood" (estado de ánimo)de las canciones

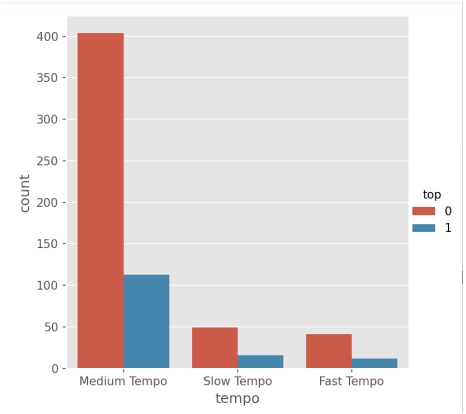


Figure 8: Vemos que hay tres tipos de tempo: Medium, Slow y Fast, y cuantas de estas han o no alcanzado el top 1.

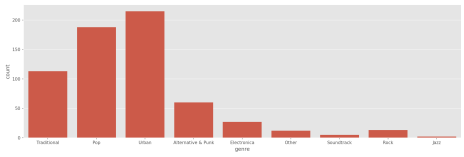


Figure 9: Entre los géneros musicales destacan Urban y Pop, seguidos de Tradicional.

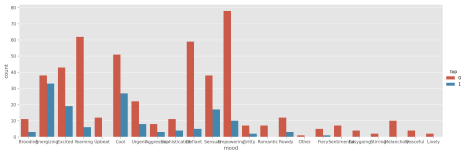


Figure 10: Visualizamos el "mood" al igual que la cuenta de cuantos de estos han o no alcanzado el top 1.

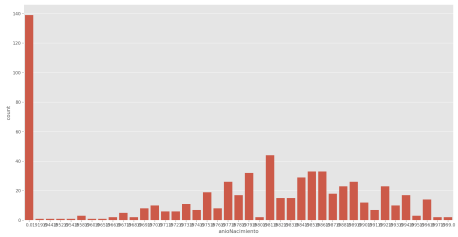


Figure 11: Notamos algo raro: en el año "cero" tenemos cerca de 140 registros..., la gráfica tiene cerca de 140 canciones de las cuales desconocemos el año de nacimiento del artista. El resto de años parecen concentrarse entre 1979 y 1994

Balanceo de Datos

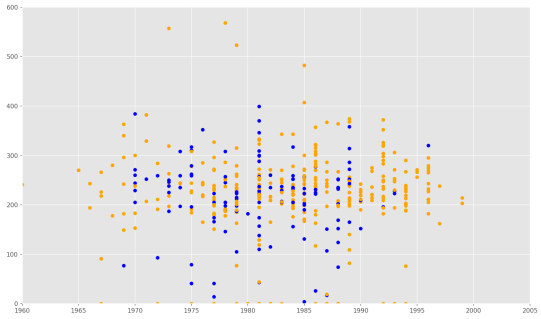


Figure 12: No vemos un patron de la relación entre Año y duración de la canción, están bastante mezclados los top 1 de los no-top

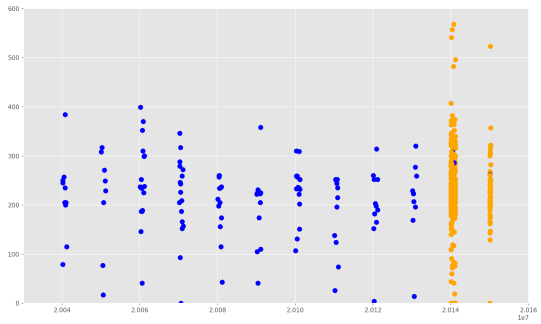


Figure 13: El conjunto de datos sigue desbalanceado, vemos las canciones que llegaron al top (en azul) de años 2004 al 2013 para sumar a los apenas 11 que lo habían logrado en 2014-2015.

Preparacion de datos

	id	title	...	artist_type	chart_date
0	0	Small Town Throwdown	...	Male	20140628
1	1	Bang Bang	...	Female	20140816
2	2	Timber	...	Mixed	20140118
3	3	Sweater Weather	...	Male	20140104
4	4	Automatic	...	Female	20140301

[5 rows x 8 columns]

Figure 14: Resultado de haber arreglado la edad de artistas.

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

Figure 15: Resultado de haber calculado el promedio de edad y asignar a los registros Nulos.

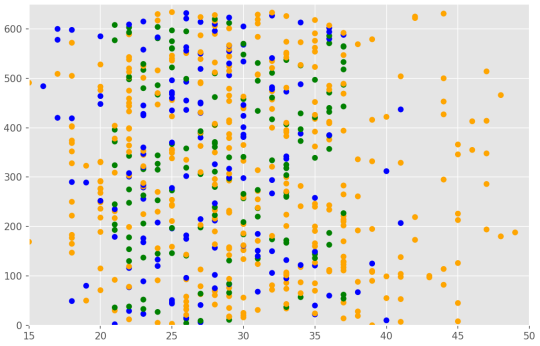


Figure 16: Visualizamos las edades que agregamos(color verde).

Como quedan los top en relación a los datos mapeados

	moodEncoded	top		
		mean	count	sum
0	0	0.000000	1	0
1	1	0.000000	8	0
2	2	0.274194	62	17
3	3	0.145631	103	15
4	4	0.136986	146	20
5	5	0.294872	156	46
6	6	0.270440	159	43

Figure 17: La mayoría del top 1 estan en los estados de ánimo 5 y 6 con 46 y 43 canciones.

	artist_typeEncoded	top		
		mean	count	sum
0	1	0.305263	95	29
1	2	0.320261	153	49
2	3	0.162791	387	63

Figure 18: Vemos repetición, pero hay mayoría en tipo 3: artistas masculinos.

	genreEncoded	top		
		mean	count	sum
0	0	0.105263	19	2
1	1	0.070000	100	7
2	2	0.008850	113	1
3	3	0.319149	188	60
4	4	0.330233	215	71

Figure 19: Los géneros con mayoría son los géneros 3 y 4 que corresponden con Urbano y Pop.

	tempoEncoded	top		
		mean	count	sum
0	0	0.226415	53	12
1	1	0.246154	65	16
2	2	0.218569	517	113

Figure 20: El tempo con más canciones exitosas es el tempo medio (2-Medio).

	durationEncoded	top		
		mean	count	sum
0	0.0	0.295775	71	21
1	1.0	0.333333	30	10
2	2.0	0.212963	108	23
3	3.0	0.202381	168	34
4	4.0	0.232143	112	26
5	5.0	0.145455	55	8
6	6.0	0.208791	91	19

Figure 21: Hay repartición del top en la relación a la duración de las canciones.

	edadEncoded	top		
		mean	count	sum
0	0.0	0.239437	71	17
1	1.0	0.309677	155	48
2	2.0	0.251701	147	37
3	3.0	0.172093	215	37
4	4.0	0.042553	47	2

Figure 22: Edad con mayoría es la tipo 1,(21 a 25 años).

Creación árbol de decisión

Max Depth	Average Accuracy
1	0.556101
2	0.556126
3	0.573413
4	0.644221
5	0.625273
6	0.629936
7	0.661285

Figure 23: vemos que la profundidad del árbol es de 7, y el score más alto 65%, está en el cuarto nivel.

Visualización árbol de decisión

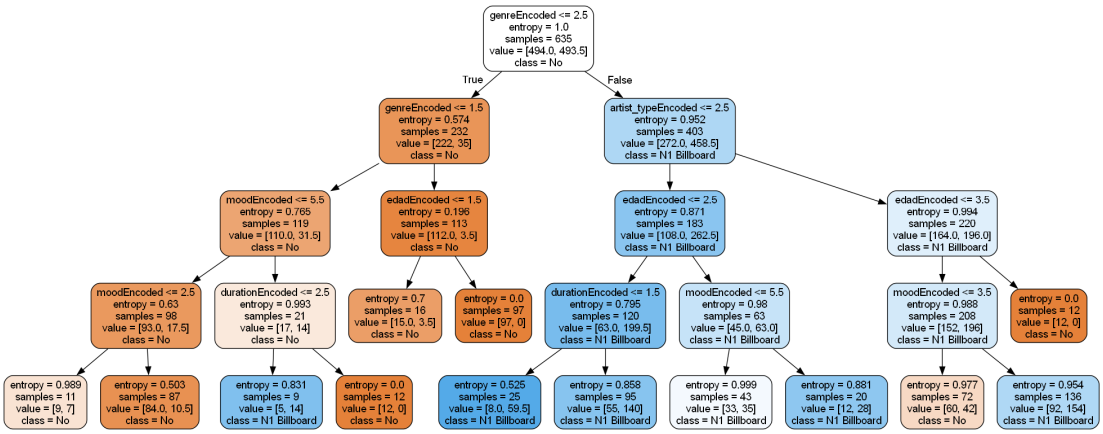


Figure 24: Árbol de Decisión

Análisis

```
68.35
```

Figure 25: La precisión del árbol tiene un valor de 64.88%.

Predicción de Canciones al Billboard 100

```
Probabilidad de Acierto: 83.73%
```

Figure 26: Nos da que Havana llegará al top 1 con una probabilidad de 83%.

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

Figure 27: La canción de Imagine Dragon NO llegará al top 1 con una certeza del 88%.

4 Conclusión

El realizar esta actividad acerca de los árboles de decisión me permitió comprender su funcionamiento e utilidad para clasificar y predecir gran cantidad de datos, destacando la importancia de todo lo que se debe de realizar antes de crear el árbol de decisión (como analizar la coherencia de los datos con los que estamos trabajando y cambiarlos en caso de que haya algo "raro" en el conjunto de datos), la selección de profundidad y la interpretación de nodos. Pude ver

la efectividad del árbol para identificar patrones en los datos y realizar predicciones claras y precisas. A parte de todo eso, el estar codificando me ayudó mucho en comprender el funcionamiento de varias funciones y métodos.

5 Referencias

Bagnato, J. (2020). Aprende Machine Learning en Español.