

Act 12: Programando Árbol de Decisión en Python

Ana Isabel Loera Gil

29 de mayo del 2025

1 Introducción

Los arboles de decisión son representaciones gráficas de posibles soluciones a una decisión basadas en ciertas condiciones, es uno de los algoritmos más utilizados en machine learning y puede realizar tareas de clasificación o regresión. Los árboles de decisión tiene un primer nodo llamado raíz y luego se descomponen el resto de los atributos de entrada en dos ramas plateando la condición que puede ser cierta o falsa. Se bifurca cada nodo en 2 y vuelve a subdividirse hasta llegar a las hojas que son los nodos finales y que equivalen a respuestas a la solución: Si/No o lo que se este clasificando.

2 Metodología

Instalar las librerías necesarias para poder ejecutar el archivo de python Las librerías usadas para está actividad son: numpy, pandas, seaborn, matplotlib, scikit-learn, ipython, pillow, graphviz Además se tuvo que instalar Graphviz del sitio oficial para poder visualizar el árbol, a continuación se muestra el link: <https://graphviz.gitlab.io/download/>

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold, cross_val_score
from IPython.display import Image as PImage
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont
from graphviz import Source
```

Lectura del archivo csv

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

Impresión de los primeros 5 registros

```
print(artists_billboard.shape)
print(artists_billboard.head())
```

Agrupación de los registros para ver cuántos alcanzaron el número uno y cuántos no

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

Visualización en forma gráfica de cuántos registros hay de tipo de artista, “mood”, tempo y género de las canciones:

```
sb.catplot(x="artist_type", data=artists_billboard, kind="count",
palette="viridis")
plt.show()
```

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

```
sb.catplot(x='genre', data=artists_billboard, kind="count", aspect=3,
palette='coolwarm')
plt.show()
```

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

Visualización de los top y no top de acuerdo a sus fechas en los Charts:

```
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()
```

Se va a sustituir los ceros de la columna “anioNacimiento” por el valor None que es nulo en Python

```
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)

```

Vamos a calcular las edades en una nueva columna `edad_en_billboard`, restando el año de aparición (los 4 primeros caracteres de `chart_date`) al año de nacimiento.

```

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)

```

Se asignaran edades aleatorias a los registros faltantes: para ello, se obtendra el promedio de edad de nuestro conjunto (`avg`) y su desviación estándar (`std`) por eso necesitábamos las edades en `None` y pedimos valores aleatorios a la función, que van desde `avg - std` hasta `avg + std`. En este caso, son edades entre 21 a 37 años.

```

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("Desviacion estandar de la edad: "+ str(age_std))
print("Intervalo para asignar edad aleatoria: "+ str(int(age_avg-age_std))+
" a "+ str(int(age_avg+age_std)))

```

Visualizacion de la informacion

```

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])
    else:
        asignar.append(colores[row['top']])

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

```

Mapeo de datos

```

artists_billboard['moodEncoded']= artists_billboard['mood'].map({
    'Energizing':6,
    'Empowering':6,
    'Cool': 5,
    'Yearning':4,
    'Excited': 5,
    'Defiant':3,
    'Sensual':2,
    'Gritty':3,
    'Sophisticated':4,
    'Aggressive':4,
    'Fiery':4,
    'Urgent':3,
    'Rowdy':4,
    'Sentimental': 4,
    'Easygoing': 1,
    'Melancholy': 4,
    'Romantic': 2,
    'Peaceful': 1,
    'Brooding': 4,
    'Upbeat': 5,
    'Stirring': 5,
    'Lively': 5,
    'Other': 0,
    '': 0}).astype(int)
artists_billboard['tempoEncoded']= artists_billboard['tempo'].map({
    'Fast Tempo':0, 'Medium Tempo':2, 'Slow Tempo':1, ''':0}).astype(int)
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)

# Mapeo de tipos de artistas
artists_billboard['artist_typeEncoded'] = artists_billboard
['artist_type'].map({
    'Female': 2,
    'Male': 3,
    'Mixed': 1,
    '': 0
}).astype(int)

# Mapeo de edad en la que llegaron al Billboard
artists_billboard.loc[artists_billboard['edad_en_billboard'] <= 21,
'edadEncoded'] = 0
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

# Mapeo de duracion de la cancion
artists_billboard.loc[artists_billboard['durationSeg'] <= 150,
'durationEncoded'] = 0
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

```

Se eliminan las columnas que no se necesitan

```

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)
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']))

```

Creación de arbol de decisión

```

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]

model = tree_model.fit(X=f_train.drop(['top'], axis=1),
y=f_train["top"])
# calculamos la precision con el segmento de validaci n
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)

# Mostramos los resultados obtenidos
df = pd.DataFrame({"MaxDepth": depth_range, "AverageAccuracy": accuracies})
df = df[["MaxDepth", "AverageAccuracy"]]
print(df.to_string(index=False))

# Crear arrays de entrenamiento y las etiquetas que indican si
llego a top o no
y_train = artists_encoded['top']
x_train = artists_encoded.drop(['top'], axis=1).values

# Crear rbol de decision con profundidad = 4
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)

# Exportar el modelo a archivo .dot
with open(r"tree1.dot", 'w') as 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', 'N1Billboard'],
        rounded=True,
        filled=True
    )

```

```

# Crear y renderizar el grafico con Graphviz
source = Source.from_file("tree1.dot")
source.render('tree1', format='png', view=True)

# Calcular la precision del modelo de arbol de decision
en el conjunto de entrenamiento
acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
print(acc_decision_tree)

Predicción de canciones

# Predicci o para el artista Camila Cabello featuring Young Thug
x_test = pd.DataFrame(columns=('top', 'moodEncoded', 'tempoEncoded',
'genreEncoded', 'artist_typeEncoded', 'edadEncoded', 'durationEncoded'))
# Valores de características para el test
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))

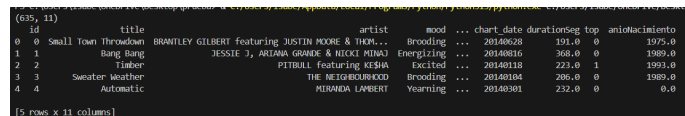
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis=1))
# Usamos y_pred[0] como indice
print(" Probabilidad de Acierto: " + str(
round(y_proba[0][y_pred[0]] * 100, 2)) + "%")

# Prediccion para el artista Imagine Dragons
# Valores de características para el test
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))
# Usamos y_pred[0] como indice
print(" Probabilidad de Acierto: " + str(
round(y_proba[0][y_pred[0]] * 100, 2)) + "%")

```

3 Resultados

A continuacion se muestran los resultados de la codificación



```

(635, 11)
  id  title  artist  mood  ... chart date durationSeg top  anioNacimiento
0  0  Small Town Throwdown  BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...  Brooding  ...  20140628  191.0  0  1975.0
1  1  Bang Bang  JESSIE J, ARIANA GRANDE & NICKI MINAJ  Energizing  ...  20140816  368.0  0  1989.0
2  2  Timber  PITBULL featuring KE$HA  Excited  ...  20140118  223.0  1  1993.0
3  3  Sweater Weather  THE NEIGHBOURHOOD  Brooding  ...  20140104  206.0  0  1989.0
4  4  Automatic  MIRANDA LAMBERT  Yearning  ...  20140301  232.0  0  0.0
[5 rows x 11 columns]

```

Figure 1: Primeros registros del archivo csv


```

top
0    494
1    141
dtype: int64

```

Figure 2: Group by del top 1

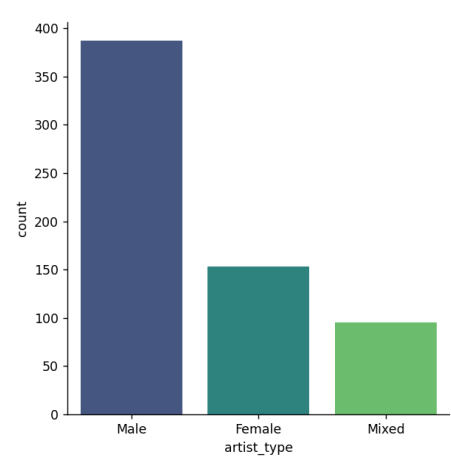


Figure 3: Gráfico de canciones por género masculino, femenino o ambos

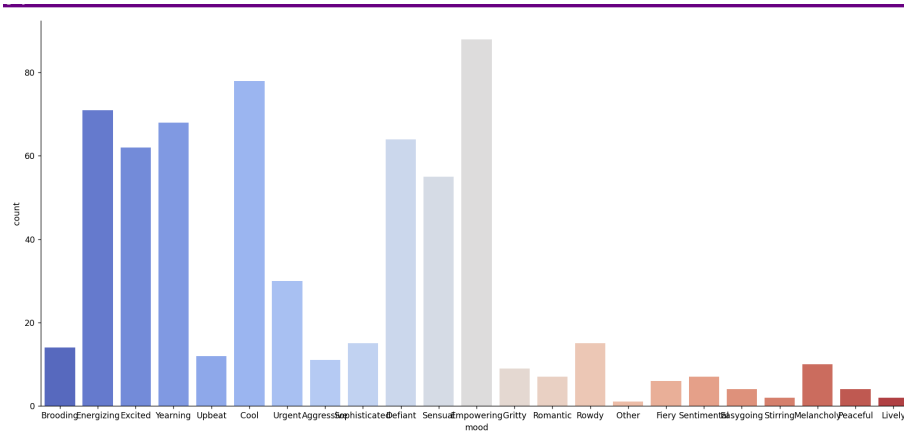


Figure 4: Gráfico de 33 tipos de mood

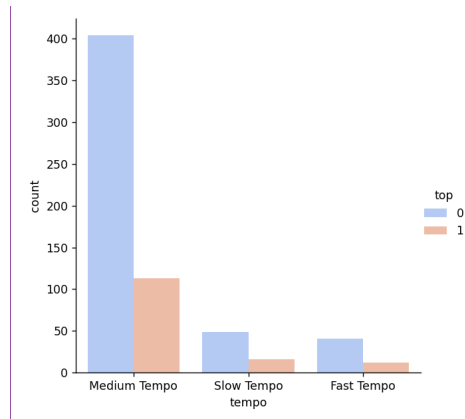


Figure 5: Gráfico de tres tipos de tempo

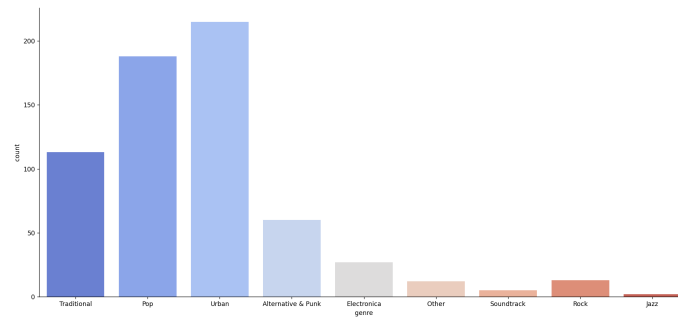


Figure 6: Gráfico por género

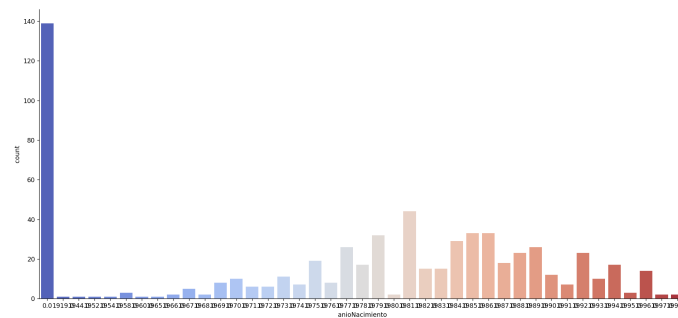


Figure 7: Gráfico por año de nacimiento del artista

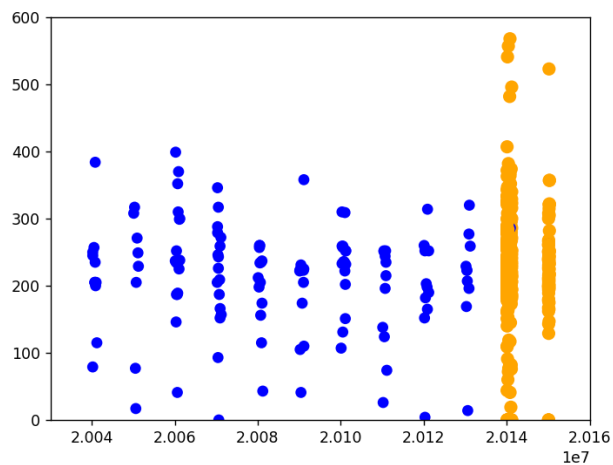


Figure 8: Gráfico canciones que llegaron al top

```

Edad promedio: 30.10282258064516
Desviacion estandar de la edad: 8.40078832861513
Intervalo para asignar edad aleatoria: 21 a 38

```

Figure 9: Información estadística de la edad del artista

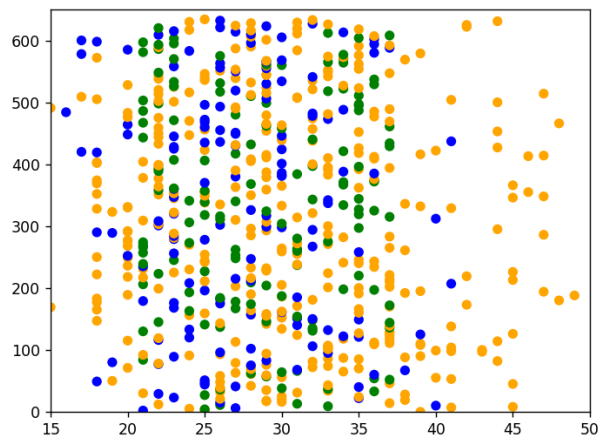


Figure 10: Gráfico canciones que llegaron al top y promedio de edades

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 11: Mood Encoded

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 12: Artist Encoded

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 13: Genre Encoded

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

Figure 14: Tempo Encoded

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 15: Duration Encoded

	edadEncoded	top	mean	count	sum
0	0.0	0.229730	74	17	
1	1.0	0.309677	155	48	
2	2.0	0.258741	143	37	
3	3.0	0.171296	216	37	
4	4.0	0.042553	47	2	

Figure 16: Edad Encoded

MaxDepth	AverageAccuracy
1	0.556101
2	0.556126
3	0.564038
4	0.650397
5	0.625347
6	0.636260
7	0.640972

Figure 17: Resultados obtenidos de la creación del árbol

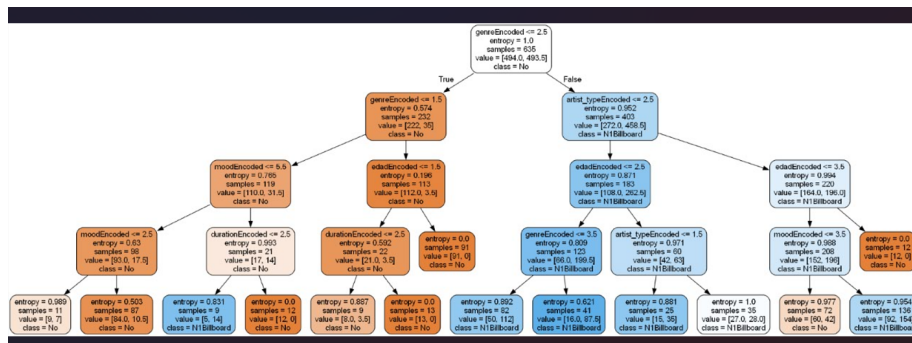


Figure 18: Árbol de decisión

68.19

Figure 19: Precisión del modelo del árbol

warnings.warn
Probabilidad de Acierto: 85.37%

Figure 20: Predicción de la canción Havana de Camila Cabello

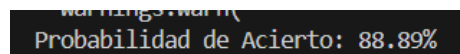


Figure 21: Predicción de la canción Beliver de Imagine Dragons

4 Conclusión

A lo largo de este proceso, he llevado a cabo diversas etapas fundamentales para la creación y validación de mi árbol de decisión. Desde la revisión y preprocesamiento de los datos hasta la conversión de valores a categorías y la generación del modelo, cada paso ha sido clave para obtener resultados significativos. Si bien el score obtenido (menor al 65%) no es particularmente alto, considero que esto se debe a la complejidad del problema planteado: predecir el número 1 del Billboard con un conjunto de datos relativamente pequeño (635 registros) y desbalanceado. Estos factores afectan la capacidad del modelo para generalizar correctamente. A pesar de estas limitaciones, el análisis realizado me ha brindado un punto de partida valioso para futuras mejoras. Posibles estrategias para optimizar el rendimiento incluyen el aumento del tamaño del conjunto de datos, el balanceo de clases y la experimentación con otros modelos de aprendizaje automático.

5 Referencias bibliograficas

Ignacio Bagnato, J. (2020). Aprende machine learning. Leanpub.