Arbol de decision en Python

[Carlos Oswaldo Gonzalez Garza]

March 31, 2025

1 Introducción

Los árboles de decisión son diagramas que muestran distintas opciones y sus posibles resultados según ciertas condiciones. Son uno de los algoritmos más usados en aprendizaje supervisado dentro del machine learning y pueden emplearse para tareas de clasificación o regresión. Un árbol de decisión organiza los datos dividiéndolos estratégicamente para mejorar la precisión de una predicción. El algoritmo elige automáticamente la mejor estructura, evitando el esfuerzo manual cuando hay muchas variables y combinaciones posibles.

2 Metodología

Se siguieron los siguientes pasos para realizar el arbol de decision

1. Para empezar importemos las librerías que utilizaremos

```
# Imports needed for the script
    import numpy as np
    import pandas as pd
    import seaborn as sb
    import matplotlib.pyplot as plt
    plt.show()
    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
```

2. Ahora veamos cuantas columnas y registros tenemos

```
artists_billboard = pd.read_csv("artists_billboard.csv")
artists_billboard.shape
```

3. Echamos un ojo a los primeros registros

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

4. Agrupamos registros para ver cuántos alcanzaron el número uno y cuantos no

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

5. Vemos cuántos registros hay de tipo de artista, "mood", tempo y género

```
sb.catplot(x="artist_type", data=artists_billboard, 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)
plt.show()
```

6. Veamos ahora que pasa al visualizar los años de nacimiento de los artistas

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

7. Visualisamos 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']])

# Gráfico de dispersión
```

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

8. Vamos a arreglar el problema de los años de nacimiento que están en cero:. Primero vamos a sustituir los ceros de la columna "anioNacimiento"por el valor None,Luego vamos a calcular las edades en una nueva columna restando el año de aparición al año de nacimiento. Y finalmente asignaremos edades aleatorias a los registros faltantes

```
# Función para manejar valores de 0 en 'anioNacimiento'
def edad_fix(anio):
    if anio == 0:
        return None
    return anio

# Función para calcular la edad
def calcula_edad(anio, cuando):
    cad = str(cuando)
    momento = cad[:4]
    if anio == 0.0:
        return None
    return int(momento) - anio

# Aplicar la función para corregir 'anioNacimiento'
artists_billboard['anioNacimiento'] = artists_billboard.apply(lambda x:
edad_fix(x['anioNacimiento']), axis=1)

# Aplicar la función para calcular la edad
artists_billboard['edad_en_billboard'] = artists_billboard.apply(lambda x:
calcula_edad(x['anioNacimiento'], x['chart_date']), axis=1)

# Calcular estadísticas de la edad
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()

# Crear una lista de edades aleatorias para los valores nulos
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std,
size=age_null_count)

# Reemplazar los valores nulos con edades aleatorias
artists_billboard.loc[artists_billboard['edad_en_billboard'].isnull(),
'edad_en_billboard'] = age_null_random_list
```

```
# Convertir la columna 'edad_en_billboard' a enteros
artists_billboard['edad_en_billboard'] = artists_billboard[
    'edad_en_billboard'].astype(int)

# Imprimir estadísticas
print("Edad Promedio: " + str(age_avg))
print("Desvío Estándar de Edad: " + str(age_std))
print("Intervalo para asignar edad aleatoria: " + str(int(age_avg - age_std))
+ " a " + str(int(age_avg + age_std)))
```

9. Podemos visualizar los valores que agregamos

```
conValoresNulos = artists_billboard['edad_en_billboard'].isnull()
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()
```

10. Vamos a transformar varios de los datos de entrada en valores categóricos.

```
        # 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'] = 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
# Mapping Song Duration
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
```

11. Revisemos en tablas cómo se reparten los top=1 en los diversos atributos mapeados.

```
artists_encoded = artists_billboard.copy()
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']))
```

12. Creamos el árbol y lo tuneamos

```
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"])
```

```
    valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
            y = f_valid["top"]) # calculamos la precision con el segmento
            de validacion
    fold_accuracy.append(valid_acc)

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

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

13. Visualización del árbol de decisión

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

# Crear Arbol 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:
     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 )

# Convertir el archivo .dot a png para poder visualizarlo
check_call(['dot','-Tpng',r'tree1.dot','-o',r'tree1.png'])
PImage("tree1.png")
```

14. Analisis

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

15. Prediccion

```
#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))
+"%")


#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))
+"%")
```

2.1 Código en Python

```
# Imports needed for the script
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
plt.show()
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
artists_billboard = pd.read_csv("artists_billboard.csv")
artists_billboard.shape
print(artists_billboard.head())
print(artists_billboard.groupby('top').size())
sb.catplot(x="artist_type", data=artists_billboard, 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)
plt.show()
sb.catplot(x='anioNacimiento',data=artists_billboard,kind="count", aspect=3)
plt.show()
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']])

# Gráfico de dispersión
plt.scatter(f1, f2, c=asignar, s=asignar2)
plt.axis([20030101, 20160101, 0, 600])
plt.show()
# Función para manejar valores de 0 en 'anioNacimiento'
def edad_fix(anio):
    if anio == 0:
        return None
    return anio

# Función para calcular la edad
def calcula_edad(anio, cuando):
    cad = str(cuando)
    momento = cad[:4]
    if anio == 0.0:
        return None
    return int(momento) - anio

# Aplicar la función para corregir 'anioNacimiento'
artists_billboard['anioNacimiento'] = artists_billboard.apply(lambda x:
```

```
edad_fix(x['anioNacimiento']), axis=1)

# Aplicar la función para calcular la edad
artists_billboard['edad_en_billboard'] = artists_billboard.apply(lambda x:
calcula_edad(x['anioNacimiento'], x['chart_date']), axis=1)

# Calcular estadísticas de la edad
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()

# Crear una lista de edades aleatorias para los valores nulos
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_std,
size=age_null_count)

# Reemplazar los valores nulos con edades aleatorias
artists_billboard.loc[artists_billboard['edad_en_billboard'].isnull(),
'edad_en_billboard'] = age_null_random_list

# Convertir la columna 'edad_en_billboard' a enteros
artists_billboard['edad_en_billboard'] =
artists_billboard['edad_en_billboard'].astype(int)

# Imprimir estadísticas
print("Edad Promedio: " + str(age_avg))
print("Desvío Estándar de Edad: " + str(age_std))
print("Intervalo para asignar edad aleatoria: " + str(int(age_avg - age_std)) + " a "
+ str(int(age_avg + age_std)))
conValoresNulos = artists_billboard['edad_en_billboard'].isnull()
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()
# Mood Mapping
artists_billboard['moodEncoded'] = artists_billboard['mood'].map( {'Energizing': 6,
```

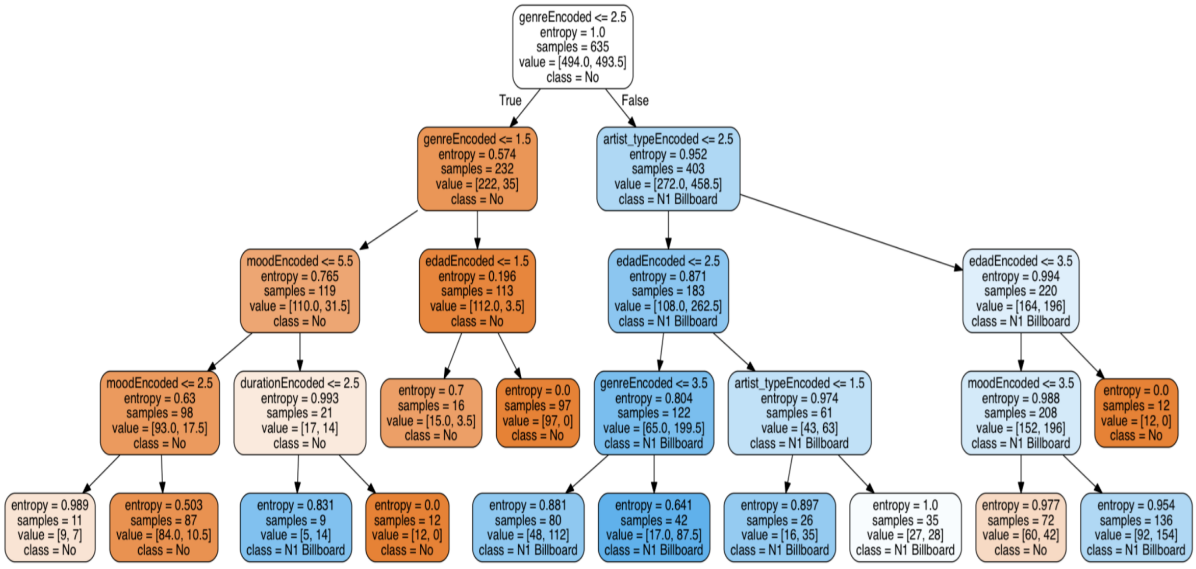
```
'Empowering': 6,
'Cool': 5,
'Yearning': 4, # anhelo, deseo, ansia
'Excited': 5, #emocionado
'Defiant': 3,
'Sensual': 2,
'Gritty': 3, #coraje
'Sophisticated': 4,
'Aggressive': 4, # provocativo
'Fiery': 4, #caracter fuerte
'Urgent': 3,
'Rowdy': 4, #ruidoso alboroto
'Sentimental': 4,
'Easygoing': 1, # 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'] = 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
# Mapping Song Duration
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
artists_encoded = artists_billboard.copy()
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']))
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"])
            valid_acc = model.score(X = f_valid.drop(['top'], axis=1),
                                    y = f_valid["top"]) # calculamos la
                                    precision con el segmento de validacion
            fold_accuracy.append(valid_acc)

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

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

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

# Crear Arbol 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:
     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 )

# Convertir el archivo .dot a png para poder visualizarlo
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)
```

```
#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))+"%")


#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




```
(635,11)
Top 0 494 1 141
```

4 Conclusión

Hemos recorrido un largo proceso para crear nuestro árbol, revisando y procesando los datos de entrada, convirtiéndolos en valores categóricos y generando el árbol. Aunque el score obtenido fue inferior al 65, lo cual no es alto, la tarea era desafiante: predecir el número 1 del Billboard con un pequeño y desbalanceado conjunto de 635 registros.