Act12

Daniel Rojas

March 2025

1 Introducción

Un árbol de decisión es un algortimo de aprendizaje supervisado perteneciente al aprendizaje profundo usado para realizar regresiones o clasificaciones.

En resumen, lo que hace es crear un nodo raíz el cuál se bifurca en dos opciones; dependiendo del valor del input, decidirá a cual de las dos hojas dirigirse. Cada nodo se dividará también en dos caminos posibles hasta llegar a los nodos finales, hojas, lo cuales darán una categoría o valor como salida dependiendo del input.

Los árboles de decisión son importantes ya que permiten realizar predicciones útiles en la gran mayoría de casos y con un bajo coste computacional. Además, el algoritmo encuentra la mejor configuración del árbol de decisión por si mismo, lo cual permite crear árboles que procesen muchos datos de entrada, sin necesidad de que un humano haga la optimización.

2 Metodología

Para realizar la práctica se tomo como referencia el artículo de Árbol de decisión del libro Aprende Machine Learning, adjuntado al final en las referencias.

La actividad consiste en predecir si la canción de un artista entrará al top de los billboard, dependiendo de diferentes factores como su tempo, edad del artista al momento del lanzamiento, género, etc. Para ello, primero se tuvo que crear una nueva columna con la edad del artista al llegar al billboard, usando las columnas de fecha de nacimiento y fecha de ingreso al billboard. Sin embargo, como las columnas de fecha de nacimiento estaban vacias, se tuvo que asignar valores aleatorias a dichos valores.

Todo el proceso se puede observar en los siguientes fragmentos de código:

```
#Librerias usadas
import numpy as np
```

```
import pandas as pd
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
```

```
#Cargar el dataset
artists_billboard = pd.read_csv('artists_billboard_fix3.csv')
```

```
#convertir valores 0 a None
def edad_fix(row):
  if row['anioNacimiento']==0:
      return None
  return row['anioNacimiento']

artists_billboard['anioNacimiento']=artists_billboard.apply(
    ↪ edad_fix, axis=1)
```

```
#Nueva columna con edad del artista al llegar al billboard
def calcula_edad(row):
    cad = str(row['chart_date'])
    year = row['anioNacimiento']
    momento = cad[:4]
    if year==0.0:
        return None
    return int(momento) - year

artists_billboard['edad_en_billboard']=artists_billboard.apply(
    ↪ calcula_edad, axis=1)
```

```
#Agregar valores aleatorios a los valores None de
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)
```

Ahora, hay que crear distintas categorías de datos para pasarlas como argumentos a nuestro árbol. Esto se realiza en las siguientes líneas de código:

```
# 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
```

```
#Quitar las columnas no categorizadas o innecesarias
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)
```

Ahora, para obtener el mejor árbol en función de la máxima profundidad permitida, se realizó un diseño de experimento provando entre árboles con 1 y 7 de profundidad para encontrar el mejor. Además, se uso la técnica de kFolds para suplementar la escasez de datos. Por último, se ajustan los pesos de cada etiqueta top a 1:3.5 en class, ya que hay 3.5 veces menos valores con el valor top = 1 que con el valor top = 0. (O sea es más común que no se llegue al top del billboard):

```
cv = KFold(n_splits=10) #9 folds para entrenar y 1 para validar
accuracies = list() #Almacena la precisi n de cada  rbol  creado
max_attributes = len(list(artists_encoded)) #N mero m ximo de
    ↪ columnas en el dataset usado
depth_range = range(1, max_attributes + 1) #Diferentes niveles de
    ↪ profundidad de los  rboles

# Testearemos la profundidad de 1 a cantidad de atributos +1
for depth in depth_range:
    fold_accuracy = [] #Precisi n de cada fold
    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))
```

Al ejecutar lo anterior se obtuvo que el mejor caso era de árbol con profundidad máxima de 4. Los valores obtenidos son los siguientes:

1. 0.556101
2. 0.556126
3. 0.564038
4. 0.650446
5. 0.603199
6. 0.620511
7. 0.636210

Ahora, para observar visualmente la estructura del mejor árbol, podemos ejecutar el siguiente código:

```
# Crear arrays de entrenamiento y las etiquetas que indican si
    ↪ lleg  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 = 4,
                              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")
```

La imagen resultante es la siguiente:

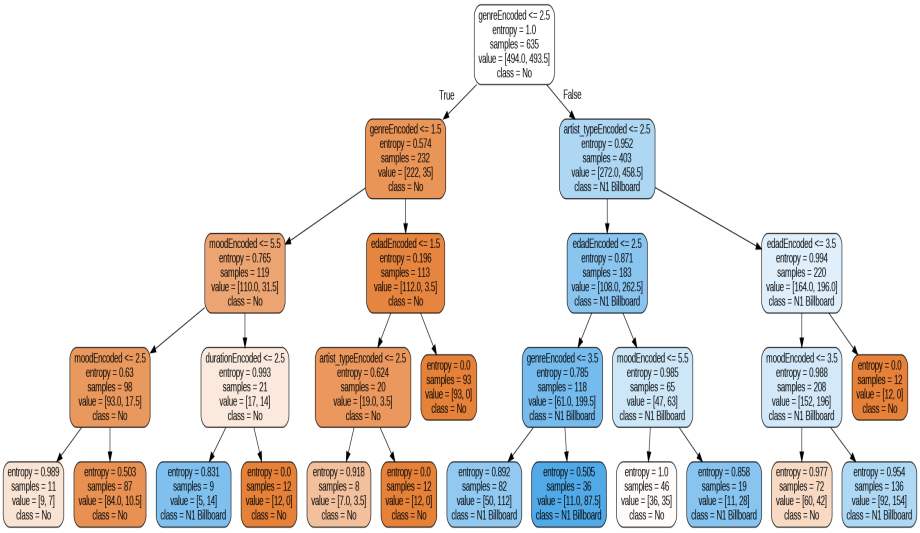

Figure 1: Árbol de 4 de profundidad

3 Resultados

Se puede observar que el mejor árbol es el que tiene una profundidad máxima de 4 niveles, con un porcentaje de precisión del 65.04%.Además, la imágen del árbol nos permite ver la forma en la que opera el árbol de decisión, observando que separa los posibles valores en función de si los valores de las features es mayor o menor a un cierto número.

4 Conclusiones

Con la actividad, aprendí de forma general el como se debe de utilizar apropiedadmente los árboles de decisión, por ejemplo, el considerar que la porporción de de clases a predecir para cambiar los valores de los pesos. Por otro lado, pude observar de forma general el funcionamiento de los árboles de decisión por medio de la imagen generada. Por último, Aprendí algunos métodos de preprocesamiento útiles para casos específicos, como el uso de valores aleatorios alrededor de la media aritmética o convertir valores continuos a categóricos.

5 Referencias

Bagnato, J. I. (2020). Aprende Machine Learning.