Actividad #12: Programando árbol de Decisión en Python

Nombre: Dayla Marely Carrizales Ortega

Matricula: 1952471

1. Introducción

Árboles de decisión es un tipo de algoritmo de aprendizaje automático supervisado utilizado por la herramienta Entrenar con AutoML y clasifica o lleva a cabo la regresión de los datos utilizando respuestas verdaderas o falsas a determinadas preguntas. La estructura resultante, cuando se visualiza, tiene la forma de un árbol con distintos tipos de nodos: de hoja, raíz e interno. El nodo raíz es el punto de partida del árbol de decisión, que, a continuación, se bifurca en nodos internos y nodos de hoja. Los nodos de hoja son las categorías o valores reales finales de clasificación. Los árboles de decisión son fáciles de comprender y se pueden explicar.

2. Metodología

Para poder realizar esta actividad se tuvieron que importar las siguientes librerías:

- **numpy y pandas** para el manejo de datos.
- **seaborn y matplotlib** para la visualización gráfica.
- **sklearn** para la creación de modelos predictivos (árboles de decisión).
- **PIL y subprocess** para la manipulación de imágenes.

Otra cosa necesaria para el funcionamiento del programa es un archivo CSV llamado artists_billboard_fix3.csv que contiene la información con la cual se va a trabajar.

Se visualiza la forma del conjunto de datos (shape) y las primeras filas (head). También, se agrupan los datos por la columna top para contar la cantidad de registros que están en el top.

Para la visualización de datos, se utilizan gráficos de tipo factorplot para observar la distribución de diversas características:

- Gráfica de conteo de canciones en el top.
- Gráfica por tipo de artista (artist_type).
- Gráfica de canciones en el top con la variable mood (estado de ánimo).

- Gráfica de canciones según el tempo y el género musical.
- Gráfica de artistas según el año de nacimiento.

En el caso del procesamiento de datos, obtenemos el filtrado de años de nacimiento inválidos o iguales a cero, cálculo de la edad del artista al momento de aparecer en la lista de Billboard y el reemplazo de valores nulos en la edad con números generados aleatoriamente dentro de un rango basado en la media y desviación estándar.

Después se realizan transformaciones categóricas de las siguientes columnas:

- **Mood (Estado de ánimo):** Se asignan valores numéricos a cada categoría de estado de ánimo.
- **Tempo:** Clasificación de ritmo en rápido, medio o lento.
- **Género:** Mapeo de géneros musicales a números enteros.
- **Tipo de Artista:** Clasificación según el género (femenino, masculino, mixto).
- **Edad al aparecer en Billboard:** Clasificación en grupos etarios.
- **Duración de la canción:** Clasificación por rangos de duración.

Se realiza un análisis de correlación para el cual se genera una matriz de correlación usando un mapa de calor para visualizar la relación entre características codificadas. Esto ayuda a identificar qué variables están más relacionadas con el hecho de que una canción esté en el top.

Luego se utiliza un modelo de Árbol de Decisión para predecir si una canción estará en el top. Acto seguido, se realiza una validación cruzada con 10 "folds" para garantizar la generalización del modelo. El modelo se ajusta con criterios de entropía y se prueban diferentes profundidades del árbol. Medimos la precisión del modelo para cada profundidad probada.

Finalmente, se calcula la precisión promedio del modelo utilizando el conjunto de datos procesados.

2.1 Código Árbol de Decisión

```
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

artists_billboard = pd.read_csv(r"artists_billboard_fix3.csv")
artists_billboard.shape
artists_billboard.head()
artists_billboard.groupby('top').size()

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

nacimientosPorAnio = artists_billboard['anioNacimiento']
len(nacimientosPorAnio[nacimientosPorAnio<=0])

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()

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

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=tamanios)
plt.axis([20030101,20160101,0,600])
plt.show()

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);
artists_billboard.head()

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)))

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()

separador = "### ### ###"
grouped11 = artists_billboard.groupby('mood').size()#.sum().reset_index()
neworder11 = grouped11.sort_values(ascending=False)
print(neworder11)
print(separador)
print("Tempos de Canción: " + str(artists_billboard['tempo'].unique()))
print(separador)
print("Tipos de Artista: " + str(artists_billboard['artist_type'].unique()))
print(separador)
grouped11 = artists_billboard.groupby('genre').size()#.sum().reset_index()
neworder11 = grouped11.sort_values(ascending=False)
print(neworder11)

# 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

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

artists_encoded.head()
artists_encoded.describe()

colormap = plt.cm.viridis
plt.figure(figsize=(12,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sb.heatmap(artists_encoded.astype(float).corr(),linewidths=0.1,vmax=1.0,
square=True, cmap=colormap, linecolor='white', annot=True)

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

3. Resultados

Los resultados obtenidos muestran que el modelo de Árbol de Decisión puede identificar características que influyen en la popularidad de una canción. Se realizaron gráficos para representar las variables más relevantes y sus correlaciones.

4. Conclusión

El uso de Árboles de Decisión permitió analizar de manera clara y estructurada los factores que pueden determinar el éxito de una canción en el ranking. La visualización de datos fue clave para entender las relaciones entre las variables.