Árboles de Decisión en Machine Learning

Alain Lobato

30 de Marzo del 2025

1 Introducción

Los árboles de decisión constituyen gráficamente la solución en posibles escenarios por condiciones, es uno de los algoritmos más utilizados en Machine Learning (ML). Inician en un nodo llamado raíz, a partir del cual se descomponen en más nodos que evalúan condiciones como ciertas o falsas. Se biparte hasta llegar a las hojas, que representan los nodos finales. Este algoritmo nos ayuda a analizar los datos para tomar la mejor decisión basada en estos datos y hacer predicciones.

2 Metodología

2.1 Espacio de trabajo

Primero clonamos nuestro repositorio de la materia en nuestra máquina, luego creamos una carpeta llamada "Árbol de Decisión". Descargamos el archivo CSV para extraer los datos y creamos un archivo llamado `decision_tree.py` donde desarrollamos nuestro código usando Visual Studio Code.

Iniciamos importando las librerías necesarias, instalando las dependencias en caso de ser necesario.

```python
# Imports needed for the script
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
from sklearn.tree import export_graphviz
from graphviz import Source
```

Figure 1: Librerias necesarias.

2.2 Carga de datos

Leemos el archivo CSV de *artists billboard* y verificamos su tamaño y los primeros 5 registros:

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

print(f"Shape: {artists_billboard.shape}")
print(artists_billboard.head())
```

Figure 2: Lectura de archivo.

```
PS C:\Users\Alain\OneDrive\Documentos\FCFM\Sem 6\IA\IntArt-EJ25-2PM\Arbol de Decision> python decision_tree.py
Shape: (635, 11)
   id                title                                             artist  ... durationSeg top anioNacimiento
0   0  Small Town Throwdown  BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...  ...       191.0   0         1975.0
1   1            Bang Bang            JESSIE J, ARIANA GRANDE & NICKI MINAJ  ...       368.0   0         1989.0
2   2              Timber                          PITBULL featuring KE$HA  ...       223.0   1         1993.0
3   3      Sweater Weather                                THE NEIGHBOURHOOD  ...       206.0   0         1989.0
4   4            Automatic                                  MIRANDA LAMBERT  ...       232.0   0            0.0

[5 rows x 11 columns]
```

Figure 3: Resultado en consola de los registros.

`Esto nos muestra: 635 registros en 11 características. Se visualizan las primeras 5 canciones registradas.`

Agrupamos las canciones que fueron top en Billboard, asignando 1 si lo fueron y 0 si no lo fueron:

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

Figure 4: Agrupamos canciones por top Billboard.

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

Figure 5: Impresion en consola.

494 canciones no han sido top Billboard y 141 sí lo han sido. Ahora analizaremos cada cancion por aspecto:

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

Figure 6: Resultado en consola de los registros.

- Tipo de artista.

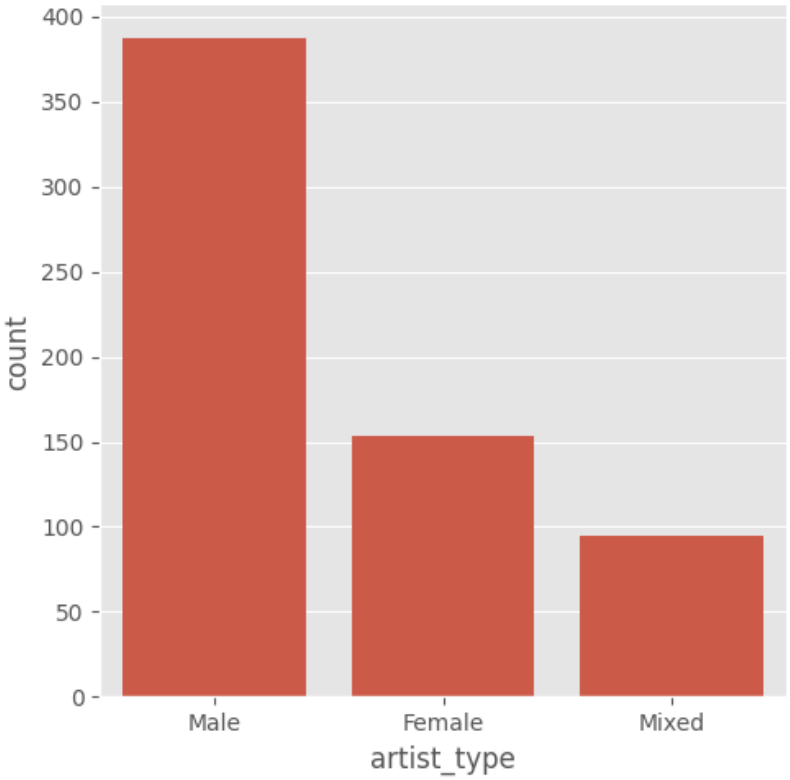

Figure 7: Resultado de analisis por tipo de artista.

- Mood.

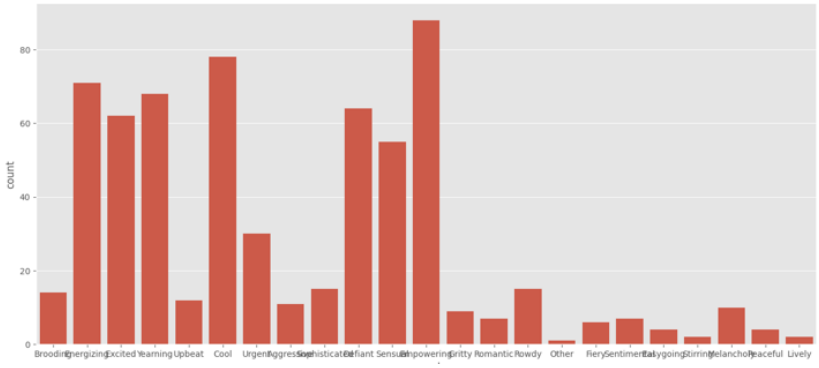

Figure 8: Resultado de analisis por mood.

- Tempo.

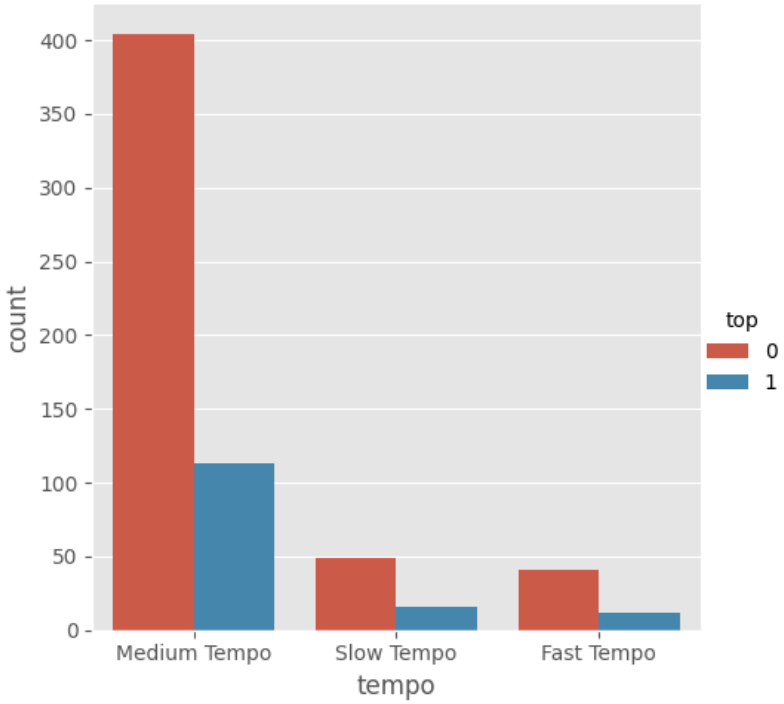

Figure 9: Resultado de analisis por tempo.

- Género.

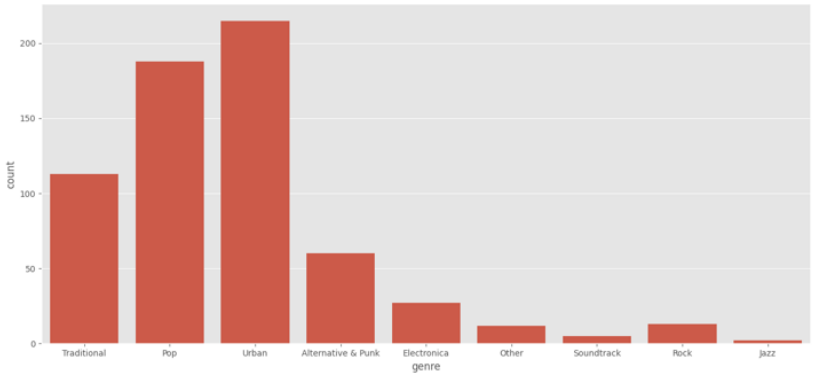

Figure 10: Resultado de analisis por género.

- Año de nacimiento.

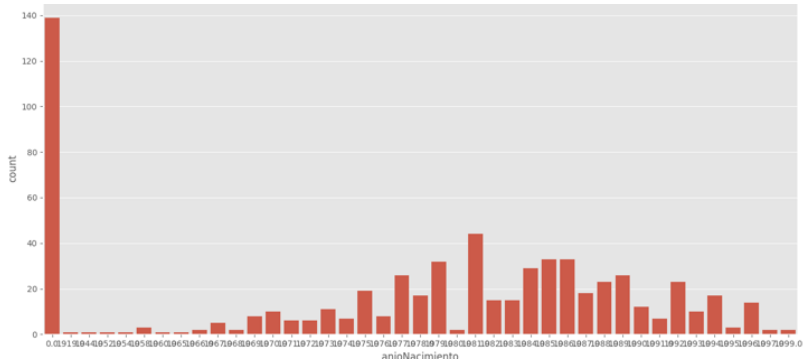

Figure 11: Resultado de analisis por año de nacimiento.

Ahora analizamos las canciones top y las que no lo fueron en función de las fechas *chart*, obteniendo los siguientes resultados:

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

asignar = []
asignar2 = []

for index, row in artists_billboard.iterrows():
    asignar.append(colores[row['top'] % 2])
    asignar2.append(tamanios[row['top'] % 2])

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

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

Figure 12: Codificacion de analisis.

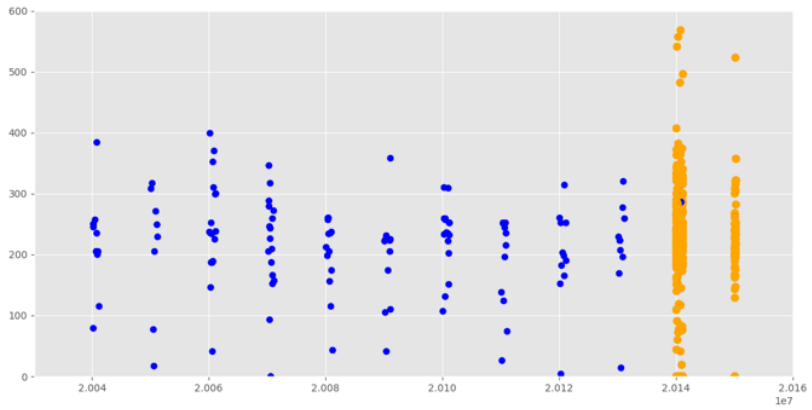

Figure 13: Resultado del grafico de analisis.

2.3 Preprocesamiento de datos

Sustituimos los años que sean 0 por un valor `None`. Calculamos la edad restando el año de aparición menos el año de nacimiento usando una función lambda, asignándola a `edad_en_billboard`.

```
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)
```

Figure 14: Codificacion para sustitucion de edades por None.

Asignamos edades aleatorias basadas en el promedio y la desviación estándar, visualizándolas en un gráfico.

```
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("DesvióStdEdad: "+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])
    else:
        asignar.append(colores[row['top']])

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

Figure 15: Asignacion de edades aleatorias.

`Edad promedio, desviación estándar e intervalo de edades a asignar.`

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

Figure 16: Analisis de lo que obtenemos en consola.

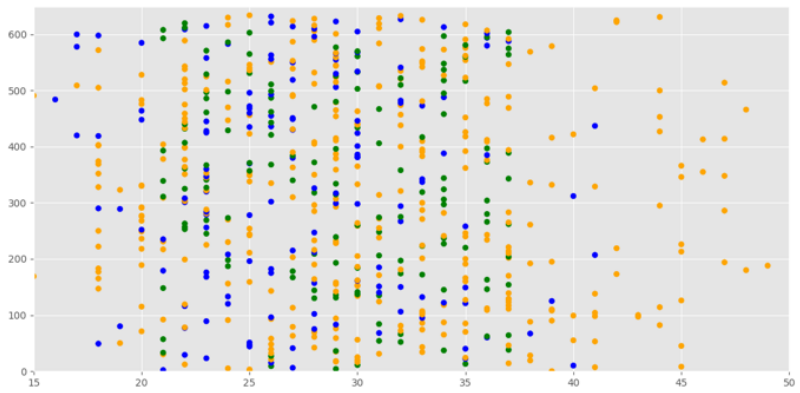

Figure 17: Resultado del grafico.

Ahora mapeamos los datos priorizando intereses y limpiamos los datos eliminando columnas no utilizadas.

```
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)
```

Figure 18: Limpieza de datos.

2.4 Construcción del árbol de decisión

Probamos el top 1 de los diferentes datos mapeados:

```
mood_table=artists_encoded[['moodEncoded', 'top']].groupby(['moodEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
print(mood_table)

artist_table= artists_encoded[['artist_typeEncoded', 'top']].groupby(['artist_typeEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
print(artist_table)

genere_table=artists_encoded[['genreEncoded', 'top']].groupby(['genreEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
print(genere_table)

tempo_table= artists_encoded[['tempoEncoded', 'top']].groupby(['tempoEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
print(tempo_table)

duration_table= artists_encoded[['durationEncoded', 'top']].groupby(['durationEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
print(duration_table)

edad_table=artists_encoded[['edadEncoded', 'top']].groupby(['edadEncoded'], as_index=False).agg(['mean', 'count', 'sum'])
print(edad_table)
```

Figure 19: Construccion de tablas con su top 1.

- Mood.

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

Figure 20: Tabla de mood con top 1.

- Artista.

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 21: Tabla de artista con top 1.

- Género.

	genreEncoded	top		
		mean	count	sum
0	0	0.088608	79	7
1	1	0.050000	40	2
2	2	0.008850	113	1
3	3	0.319149	188	60
4	4	0.330233	215	71

Figure 22: Tabla de genero con top 1.

- Tempo.

	tempoEncoded	top		
		mean	count	sum
0	0	0.222047	635	141

Figure 23: Tabla de tempo con top 1.

- Duración.

	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 24: Tabla de duración con top 1.

- Edad.

	edadEncoded	top		
		mean	count	sum
0	0.0	0.253731	67	17
1	1.0	0.303797	158	48
2	2.0	0.256944	144	37
3	3.0	0.168950	219	37
4	4.0	0.042553	47	2

Figure 25: Tabla de edad con top 1.

Finalmente, creamos el árbol de decisión con `sklearn`, definiendo los parámetros necesarios. Realizamos pruebas para determinar la profundidad ideal del árbol:

```
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)
```

Figure 26: Construccion de arbol de decision.

```
#Testearemos la profundidad de 1 acantidad 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"])#calculamoslaprecisionconelsegmentode validacion
    fold_accuracy.append(valid_acc)


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

Figure 27: Pruebas.

MaxDepth	AverageAccuracy
1	0.556101
2	0.556126
3	0.564038
4	0.645759
5	0.615848
6	0.625248
7	0.647247

Figure 28: Tabla para visualizar el valor optimo del Max Depth.

`Profundidad óptima encontrada: 7.`

Visualizamos el árbol configurado y verificamos su precisión.

```
# Usar graphviz para convertir el archivo .dot a una imagen .png
Source.from_file("tree1.dot").render("tree1", format="png")

acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
print("Precisión del árbol: ", acc_decision_tree)
```

Figure 29: Verificamos la precision.

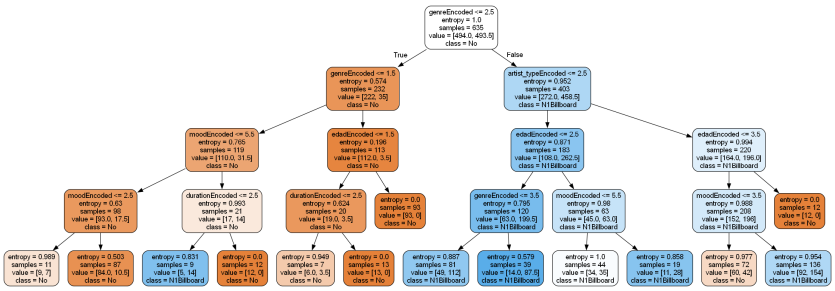

Figure 30: Arbol de decision.

```
Precisión del árbol:  73.54
```

Figure 31: Precision del arbol.

3 Resultados

Para probar el árbol de decisión, evaluamos dos canciones: *Havana* de Camila Cabello (top) y *Believer* de Imagine Dragons (no top):

- **Havana – Camila Cabello**: Resultado del modelo.

```
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
proba=y_proba[0][y_pred]*100
print("Prediccion: " + str(y_pred))
print("Probabilidad de Acierto Havana: " + str(round(proba[0], 2))+"%")
```

Figure 32: Codificacion para analizar la cancion.

```
Prediccion: [1]
Probabilidad de Acierto Havana: 90.1%
```

Figure 33: Resultado.

- **Believer – Imagine Dragons**: Resultado del modelo.

```
y_proba = decision_tree.predict_proba(x_test.drop(['top'], axis = 1))
proba2 = y_proba[0][y_pred]* 100
print("Prediccion: " + str(y_pred))
print("Probabilidad de Acierto Believer: " + str(round(proba2[0], 2))+"%")
```

Figure 34: Codificacion para analizar la cancion.

```
Prediccion: [0]
Probabilidad de Acierto Believer: 65.0%
```

Figure 35: Resultado.

El modelo predice que la canción *Havana* de Camila Cabello es un top 1 con una probabilidad de 90.1% y que *Believer* de Imagine Dragons no es un top 1 con una probabilidad de 65%.

4 Conclusión

Se presentaron inconvenientes con la codificación debido a errores en el código original y un valor incorrecto de `max_depth` (4 en lugar de 7). Se instalaron las paqueterías necesarias y se solucionó un problema con Graphviz que no identificaba el `PATH`. A pesar de estos desafíos, modificando y limpiando los datos, logramos obtener un resultado favorable. De cualquier manera durante esta actividad se aprendió a usar el algoritmo de árboles de decisión, su construcción y la importancia de la limpieza de datos.

5 Referencias

- Materiales de clase (2025). UANL.
- Bagnato J (2019). *Aprende Machine Learning.* Leanpub.