Árbol de decisión

¿Qué es un árbol de decisión?

Los árboles de decisión son representaciones gráficas de posibles soluciones a una decisión basadas en ciertas condiciones, es uno de los algoritmos de aprendizaje supervisado más utilizados en machine learning y pueden realizar tareas de clasificación o regresión. La comprensión de su funcionamiento suele ser simple y a la vez muy potente.

Utilizamos mentalmente estructuras de árbol de decisión constantemente en nuestra vida diaria sin darnos cuenta:

¿Llueve? => lleva paraguas. ¿Soleado? => lleva gafas de sol. ¿estoy cansado? => toma café. (decisiones del tipo IF THIS THEN THAT)

Los árboles de decisión tienen un primer nodo llamado raíz y luego se descomponen el resto de atributos de entrada en dos ramas planteando una condición que puede ser cierta o falsa. Se bifurca cada nodo en 2 y vuelven a subdividirse hasta llegar a las hojas que son los nodos finales y que equivalen a respuestas a la solución: Si/No, Comprar/Vender, o lo que sea que estemos clasificando.

Ejemplo:

¿Animal ó vegetal? -Animal
¿Tiene cuatro patas? -Si
¿Hace guau? -Si
-> Es un perro!

¿Qué necesidad hay de usar el Algoritmo de Árbol?

Supongamos que tenemos atributos como Género con valores “hombre ó mujer” y edad en rangos: “menor de 18 ó mayor de 18” para tomar una decisión. Podríamos crear un árbol en el que dividamos primero por género y luego subdividir por edad. Ó podría ser al revés: primero por edad y luego por género. El algoritmo es quien analizando los datos y las salidas decidirá la mejor forma de hacer las divisiones entre nodos. Tendrá en cuenta de qué manera lograr una predicción con mayor probabilidad de acierto. Parece sencillo, no? Pensemos que si tenemos 10 atributos de entrada cada uno con 2 o más valores posibles, las combinaciones para decidir el mejor árbol serían cientos ó miles… Esto ya no es un trabajo para hacer artesanalmente. Y ahí es donde este algoritmo cobra importancia, pues él nos devolverá el árbol óptimo para la toma de decisión más acertada desde un punto de vista probabilístico.

¿Cómo funciona un árbol de decisión?

Para obtener el árbol óptimo y valorar cada subdivisión entre todos los árboles posibles y conseguir el nodo raíz y los subsiguientes, el algoritmo deberá medir de alguna manera las predicciones logradas y valorarlas para comparar de entre todas y obtener la mejor. Para medir y valorar, utiliza diversas funciones, siendo las más conocidas y usadas los “Índice gini” y “Ganancia de información” que utiliza la denominada “entropía“. La división de nodos

continuará hasta que lleguemos a la profundidad máxima posible del árbol ó se limiten los nodos a una cantidad mínima de muestras en cada hoja.

Índice Gini:

Se utiliza para atributos con valores continuos (precio de una casa). Esta función de coste mide el “grado de impureza” de los nodos, es decir, cuán desordenados o mezclados quedan los nodos una vez divididos. Deberemos minimizar ese GINI index.

Ganancia de información:

Se utiliza para atributos categóricos (como en hombre/mujer). Este criterio intenta estimar la información que aporta cada atributo basado en la “teoría de la información“. Para medir la aleatoriedad de incertidumbre de un valor aleatorio de una variable “X” se define la Entropía. Al obtener la medida de entropía de cada atributo, podemos calcular la ganancia de información del árbol. Deberemos maximizar esa ganancia.

Ejemplo en Python:

Para empezar importamos las librerías que utilizaremos y revisemos sus atributos de entrada:

```
# 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

artists_billboard = pd.read_csv(“artists_billboard_fix3.csv”)

artists_billboard.head()
```

	id	title	artist	mood	tempo	genre	artist_type	chart_date	durationSeg	top	anioNacimiento
0	0	Small Town Throwdown	BRANTLEY GILBERT featuring JUSTIN MOORE & THOM...	Brooding	Medium Tempo	Traditional	Male	20140628	191.0	0	1975.0
1	1	Bang Bang	JESSIE J, ARIANA GRANDE & NICKI MINAJ	Energizing	Medium Tempo	Pop	Female	20140816	368.0	0	1989.0
2	2	Timber	PITBULL featuring KE$HA	Excited	Medium Tempo	Urban	Mixed	20140118	223.0	1	1993.0
3	3	Sweater Weather	THE NEIGHBOURHOOD	Brooding	Medium Tempo	Alternative & Punk	Male	20140104	206.0	0	1989.0
4	4	Automatic	MIRANDA LAMBERT	Yearning	Medium Tempo	Traditional	Female	20140301	232.0	0	0.0

Vemos que tenemos: Titulo de la canción, artista, “mood”, género, Tipo de artista, fecha en que apareció en el billboard (por ejemplo 20140628 equivale al 28 de junio de 2014), la columna TOP será nuestra etiqueta, target ó label en la que aparece 1 si llegó al número uno de Billboard ó 0 si no lo alcanzó y el año de Nacimiento del artista. Vemos que muchas de las columnas contienen información categórica. La columna durationSeg contiene la duración en segundos de la canción, siendo un valor continuo pero que nos convendrá pasar a categórico más adelante.

Primero, agrupamos registros para ver cuántos alcanzaron el número uno y cuántos no:

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

```
top
0 494
1 141
```

Preparamos los datos

Vamos a arreglar el problema de los años de nacimiento que están en cero. Realmente el “feature” o característica que queremos obtener es : “sabiendo el año de nacimiento del cantante, calcular qué edad tenía al momento de aparecer en el Billboard”. Por ejemplo un artista que nació en 1982 y apareció en los charts en 2012, tenía 30 años.

Primero vamos a sustituir los ceros de la columna “anioNacimiento”por el valor None.

```
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)
```

Luego 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. En las filas que estaba el año en None, tendremos como resultado edad None.

```
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(lam
bda x: calcula_edad(x['anioNacimiento'],x['chart_date']), axis=1)
```

Y finalmente asignaremos edades aleatorias a los registros faltantes: para ello, obtenemos el promedio de edad de nuestro conjunto y su desvío estándar (por eso necesitábamos las edades en None) y pedimos valores random a la función que van desde avg – std hasta avg + std. En nuestro caso son edades de 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_billboar
d']), '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)))
```

Mapeo de Datos

Vamos a transformar varios de los datos de entrada en valores categóricos. Las edades, las separamos en: menor de 21 años, entre 21 y 26, etc. las duraciones de canciones también, por ej. entre 150 y 180 segundos, etc. Para los estados de ánimo (mood) agrupé los que eran similares.

El Tempo que puede ser lento, medio o rápido queda mapeado: 0-Rapido, 1-Lento, 2-Medio (por cantidad de canciones en cada tempo: el Medio es el que más tiene)

```
# 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
```

Finalmente obtenemos un nuevo conjunto de datos llamado artists_encoded con el que tenemos los atributos definitivos para crear nuestro árbol. Para ello, quitamos todas las columnas que no necesitamos con “drop”:

```
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)
```

Buscamos la profundidad para nuestro árbol de decisión

Ya casi tenemos nuestro árbol. Antes de crearlo, vamos a buscar cuántos niveles de profundidad le asignaremos. Para ello, aprovecharemos la función de KFold que nos ayudará a crear varios subgrupos con nuestros datos de entrada para validar y valorar los árboles con diversos niveles de profundidad. De entre ellos, escogeremos el de mejor resultado.

Creamos el árbol y lo tuneamos

Para crear el árbol utilizamos tree.DecisionTreeClasifier pues buscamos un árbol de clasificación (no de Regresión). Lo configuramos con los parámetros:

criterion=entropy ó podría ser gini, pero utilizamos entradas categóricas
min_samples_split=20 se refiere a la cantidad mínima de muestras que debe tener un nodo para poder subdividir.
min_samples_leaf=5 cantidad mínima que puede tener una hoja final. Si tuviera menos, no se formaría esa hoja y "subiría" un nivel, su antecesor.
class_weight={1:3.5} IMPORTANTÍSIMO: con esto compensamos los desbalances que hubiera. En nuestro caso, como venía diciendo anteriormente, tenemos menos etiquetas de tipo top=1 (los artistas que llegaron al número 1 del ranking). Por lo tanto, le asignamos 3.5 de peso a la etiqueta 1 para compensar. El valor sale de dividir la cantidad de top=0 (son 494) con los top=1 (son 141).
NOTA: estos valores asignados a los parámetros fueron puestos luego de prueba y error (muchas veces visualizando el árbol, en el siguiente paso y retrocediendo a este).

```
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))
```

Visualización del árbol de decisión

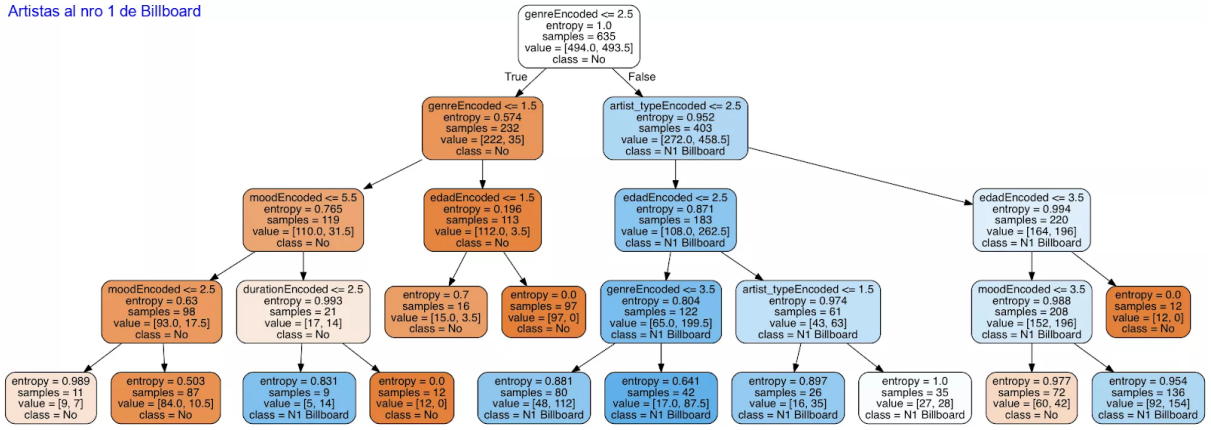
Asignamos los datos de entrada y los parámetros que configuramos anteriormente con 4 niveles de profundidad. Utilizaremos la función de export_graphviz para crear un archivo de extensión .dot que luego convertiremos en un gráfico png para visualizar el árbol.

```
# 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")
```



Conclusiones y análisis del árbol

En la gráfica vemos, un nodo raíz que hace una primer subdivisión por género y las salidas van a izquierda por True que sea menor a 2.5, es decir los géneros 0, 1 y 2 (eran los que menos top=1 tenían) y a derecha en False van los géneros 3 y 4 que eran Pop y Urban con gran cantidad de usuarios top Billboard.

En el segundo nivel vemos que la cantidad de muestras queda repartida en 232 y 403 respectivamente.

A medida que bajamos de nivel veremos que los valores de entropía se aproximan más a 1 cuando el nodo tiene más muestras top=1 (azul) y se acercan a 0 cuando hay mayoría de muestras Top=0 (naranja).

En los diversos niveles veremos divisiones por tipo de artista , edad, duración y mood. También vemos algunas hojas naranjas que finalizan antes de llegar al último nivel: esto es porque alcanzan un nivel de entropía cero, o porque quedan con una cantidad de muestras menor a nuestro mínimo permitido para hacer split (20).

Predicción de Canciones al Billboard 100

Vamos a testear nuestro árbol con 2 artistas que entraron al billboard 100 en 2017: Camila Cabello que llegó al numero 1 con la Canción Havana y Imagine Dragons con su canción Believer que alcanzó un puesto 42 pero no llegó a la cima.

```
#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]*
100, 2))+"%")
```

Nos da que Havana llegará al top 1 con una probabilidad del 83%. Nada mal…

```
#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]*
100, 2))+"%")
```

Nos da que la canción de Imagine Dragons NO llegará con una certeza del 88%. Otro acierto.

Veamos los caminos tomados por cada una de las canciones: