CLASIFICACIÓN MULTIPLE - POPULARIDAD

Integrantes de equipo:

- Julio César Choquehuayta Quenta 2018-119025
- Carlos Enrique Yufra Loza 2019-119051
- Pablo Moisés Aro Galindo 2019-119034
- Carlos Manuel Azañero Otoya 2015-119026
- Jose Angel Castro Caceres 2017-119054

DESCRIPCION DEL PROBLEMA

La industria de la música es bastante popular en la actualidad, lo que escuchan las personas cambia con el tiempo, ya sea por tendencias o moda, mucho de lo que determina su popularidad suele ser subjetivo o circunstancial. Sin embargo tomando en cuenta datos técnicos o variables presentes en la musica, se tendrá como objetivo el determinar si es popular o no.

Este proyecto se enfocará en un problema de clasificacion múltiple, el cual hara uso de una dataset de Spotify de pistas en un rango de 125 géneros diferentes. Cada pista tiene algunas funciones de audio asociadas. Los datos están en formato CSV, que es tabular y se puede cargar rápidamente.

Link del Dataset: https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset?resource=download

ANALISIS EXPLORATORIO

Primero, nosotros necesitamos tener la dataset descargada y almacenada en una carpeta en drive para su próximo uso, ademas de importar las librerías necesarias.

```
import numpy as np
import pandas as pd
from keras import layers,models, optimizers
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
```

Nosotros cargamos la dataset usando pandas read_csv, junto con el link de donde esta nuestra data en drive

```
# Obtenemos el dataset de las Canciones de Spotify y lo almacenamos en un DataFrame
df = pd.read_csv("https://drive.google.com/uc?id=1gAkvuBkYAEKCdf9CYqePni8TacqkwWt3")
df
```

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	dur
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	
...	...	...	...	...	...	...	
113995	113995	2C3TZjDRiAzdyViavDJ217	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Sleep My Little Boy	21	
113996	113996	1hIz5L4IB9hN3WRYPOCGPw	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Water Into Light	22	
113997	113997	6x8ZfSoqDjuNa5SVP5QjvX	Cesária Evora	Best Of	Miss Perfumado	22	
113998	113998	2e6sXL2bYv4bSz6VTdnfLs	Michael W. Smith	Change Your World	Friends	41	
113999	113999	2hETkH7cOfqmz3LqZDHZf5	Cesária Evora	Miss Perfumado	Barbincor	22	

114000 rows × 21 columns

Exploramos un poco la dataframe a usar, usando algunas sentencias para visualizar mejor las columnas, tipos de datos y datos únicos

```
1 # Visualizamos las primeras 12 columnas
2 df.iloc[:,:12]
```

	Unnamed: 0	track_id	artists	album_name	track_name	popula
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	
...	...	...	...	...	...	
113995	113995	2C3TZjDRiAzdyViavDJ217	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Sleep My Little Boy	
113996	113996	1hIz5L4IB9hN3WRYPOCGPw	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Water Into Light	
113997	113997	6x8ZfSoqDjuNa5SVP5QjvX	Cesária Evora	Best Of	Miss Perfumado	
113998	113998	2e6sXL2bYv4bSz6VTdnfLs	Michael W. Smith	Change Your World	Friends	
113999	113999	2hETkH7cOfqmz3LqZDHZf5	Cesária Evora	Miss Perfumado	Barbincor	

```
1 # Visualizamos el resto de columnas
2 df.iloc[:,12:]
```

	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature
0	0	0.1430	0.0322	0.000001	0.3580	0.7150	87.917	4
1	1	0.0763	0.9240	0.000006	0.1010	0.2670	77.489	4
2	1	0.0557	0.2100	0.000000	0.1170	0.1200	76.332	4
3	1	0.0363	0.9050	0.000071	0.1320	0.1430	181.740	3
4	1	0.0526	0.4690	0.000000	0.0829	0.1670	119.949	4
...	...	...	...	...	...	...	...	...
113995	1	0.0422	0.6400	0.928000	0.0863	0.0339	125.995	5
113996	0	0.0401	0.9940	0.976000	0.1050	0.0350	85.239	4
113997	0	0.0420	0.8670	0.000000	0.0839	0.7430	132.378	4
113998	1	0.0297	0.3810	0.000000	0.2700	0.4130	135.960	4
113999	0	0.0725	0.6810	0.000000	0.0893	0.7080	79.198	4

114000 rows × 9 columns

```
# Visualizamos el tipo de dato de columna
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 21 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   Unnamed: 0        114000 non-null  int64  
 1   track_id          114000 non-null  object 
 2   artists           113999 non-null  object 
 3   album_name        113999 non-null  object 
 4   track_name        113999 non-null  object 
 5   popularity        114000 non-null  int64  
 6   duration_ms       114000 non-null  int64  
 7   explicit          114000 non-null  bool   
 8   danceability      114000 non-null  float64
 9   energy            114000 non-null  float64
 10  key               114000 non-null  int64  
 11  loudness          114000 non-null  float64
 12  mode              114000 non-null  int64  
 13  speechiness       114000 non-null  float64
 14  acousticness      114000 non-null  float64
 15  instrumentalness  114000 non-null  float64
 16  liveness          114000 non-null  float64
 17  valence           114000 non-null  float64
 18  tempo             114000 non-null  float64
 19  time_signature    114000 non-null  int64  
 20  track_genre       114000 non-null  object 
dtypes: bool(1), float64(9), int64(6), object(5)
memory usage: 17.5+ MB
```

Cuando se hace el llamado a df.info, este imprimira la siguiente informacion:

- El numero de filas y columnas en la dataframe
- El nombre de cada columna, su tipo de dato, y el numero de non-null values en la columna
- El numero total de non.null values en la dataframe
- La memoria de uso en la dataframe

```
# number of unique values in our data set.
df.nunique()
```

```
Unnamed: 0          114000
track_id             89741
artists              31437
album_name           46589
track_name           73608
popularity             101
duration_ms          50697
explicit                 2
danceability          1174
energy                2083
key                     12
loudness              19480
mode                      2
speechiness            1489
acousticness           5061
instrumentalness       5346
liveness               1722
```

```
valence              1790
tempo               45653
time_signature          5
track_genre           114
dtype: int64
```

PRE-PROCESAMIENTO DE LOS DATOS

```
# Convertimos la columna 'popularity' en 'popularity_class'(0:baja, 1:media, 2:alta)
df['popularity_class'] = 0
df.loc[df['popularity'] > 30, 'popularity_class'] = 1
df.loc[df['popularity'] > 70, 'popularity_class'] = 2
```

```
# Extraemos una muestra de 100 000 registros aleatorios
df_sample = df.sample(100000)

# Filtramos las entradas y la salida que usaremos para el modelo
columns = ['danceability','energy','loudness','speechiness','acousticness','instrumentalness','valence'
df_columns = df_sample[columns]
df_columns.nunique() # visualizamos la cantidad de valores únicos de las columnas
```

```
danceability         1146
energy               2028
loudness            18771
speechiness          1475
acousticness         4987
instrumentalness     5322
valence              1766
tempo               42506
popularity_class        3
dtype: int64
```

```
# Visualizamos algunos registros filtrados por el valor de 'popularity_class'
df_columns.query('popularity_class == 2')
```

	danceability	energy	loudness	speechiness	acousticness	instrumentalness	valence	tempo
106005	0.713	0.452	-9.638	0.0295	0.841000	0.003800	0.235	102.963
91518	0.338	0.340	-12.049	0.0339	0.580000	0.003200	0.197	82.433

```
# Visualizamos algunas estadísticas de nuestros datos
df_columns.describe()
```

	danceability	energy	loudness	speechiness	acousticness	instrumentalness	
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100
mean	0.566870	0.641210	-8.265175	0.084553	0.315162	0.156101	
std	0.173493	0.251575	5.028899	0.105449	0.332646	0.309659	
min	0.000000	0.000000	-49.531000	0.000000	0.000000	0.000000	
25%	0.456000	0.472000	-10.019000	0.035900	0.017000	0.000000	
50%	0.580000	0.685000	-7.013000	0.048900	0.169000	0.000041	
75%	0.694000	0.853000	-5.010000	0.084500	0.598000	0.049000	
max	0.985000	1.000000	4.532000	0.965000	0.996000	1.000000	

```
# Pasamos los datos del DF a un arreglo
data = df_columns.values

# Separamos las características de las etiquetas
x_data = data[:, :-1]
y_data = data[:, -1]

print("X:")
print(x_data)
print("y:")
print(y_data)
```

```
X:
[[ 7.68000e-02  1.31000e-01 -2.31530e+01 ...  7.97000e-01  3.56000e-02
   8.07650e+01]
 [ 3.73000e-01  8.57000e-01 -3.54600e+00 ...  1.38000e-03  2.35000e-01
   1.37084e+02]
 [ 7.37000e-01  7.05000e-01 -6.23100e+00 ...  0.00000e+00  5.57000e-01
   8.64010e+01]
 ...
 [ 6.29000e-01  8.37000e-01 -5.67700e+00 ...  4.39000e-01  7.72000e-01
   1.25013e+02]
 [ 2.78000e-01  1.94000e-01 -1.77770e+01 ...  9.20000e-01  2.21000e-01
   6.93680e+01]
 [ 7.20000e-01  7.38000e-01 -1.10840e+01 ...  9.27000e-01  5.48000e-01
   9.59950e+01]]
y:
[0. 1. 0. ... 0. 1. 0.]
```

```
# Verificamos las dimensiones de los arreglos
print("X:")
```

```
print(x_data.shape)
print("y:")
print(y_data.shape)
```

```
X:
(100000, 8)
y:
(100000,)
```

```
# Obtenemos la media y la desviación estándar de cada característica
x_mean = x_data.mean(axis = 0)
x_std = x_data.std(axis = 0)

# Normalizamos las características del modelo y convertimos a float32
x_data = (x_data - x_mean) / x_std
x_data= x_data.astype(np.float32)

# Verificamos los datos normalizados
x_data
```

```
array([[-2.82474   , -2.0280714 , -2.960469  , ...,  2.0697055 ,
        -1.6917396 , -1.3791927 ],
       [-1.1174587 ,  0.8577615 ,  0.93841594, ..., -0.49965143,
        -0.92200196,  0.5002919 ],
       [ 0.9806183 ,  0.2535651 ,  0.40449914, ..., -0.50410795,
         0.32100457, -1.1911074 ],
       ...,
       [ 0.35811195,  0.77826196,  0.514663  , ...,  0.9135886 ,
         1.1509624 ,  0.097457  ],
       [-1.6650337 , -1.7776479 , -1.8914424 , ...,  2.466919  ,
        -0.9760457 , -1.7595347 ],
       [ 0.8826312 ,  0.3847393 , -0.5605281 , ...,  2.4895246 ,
         0.28626215, -0.87093526]], dtype=float32)
```

DISEÑO DE LA ARQUITECTURA Y ENTRENAMIENTO DEL MODELO DE RED NEURONAL MULTICAPA

```
# Separamos aleatoriamente los datos de entrenamiento(80%) y de prueba(20%)
X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.2, random_state=20)
```

```
# Definimos el modelo
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(8,)))
model.add(layers.Dropout(0.125))
model.add(layers.Dense(512, activation='relu'))
# model.add(layers.Dropout(0.125))
model.add(layers.Dense(512, activation='relu'))
# model.add(layers.Dropout(0.125))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))

# Compilamos el modelo
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics='accuracy')

# Entrenamos el modelo
```

```
16 history = model.fit(X_train, y_train, validation_split=0.2, epochs=100, batch_size=1024)
```

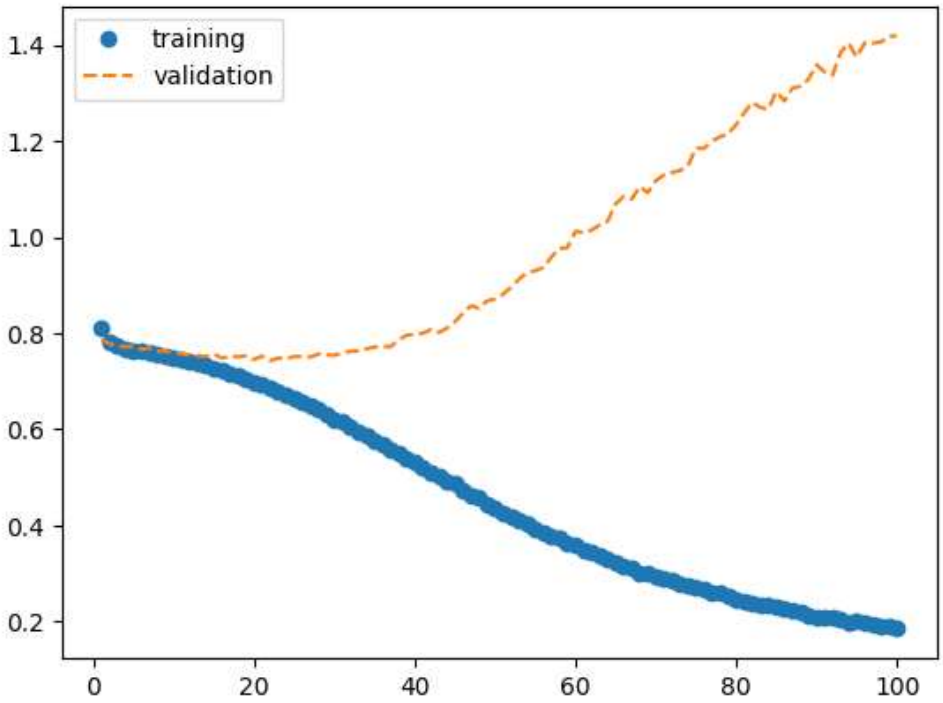
```
3 [==============================] - 0s 5ms/step - loss: 0.2855 - accuracy: 0.8780 - val_loss: 1.134
h 73/100
3 [==============================] - 0s 5ms/step - loss: 0.2766 - accuracy: 0.8836 - val_loss: 1.138
h 74/100
3 [==============================] - 0s 5ms/step - loss: 0.2738 - accuracy: 0.8863 - val_loss: 1.148
h 75/100
3 [==============================] - 0s 5ms/step - loss: 0.2722 - accuracy: 0.8846 - val_loss: 1.18
h 76/100
3 [==============================] - 0s 5ms/step - loss: 0.2676 - accuracy: 0.8881 - val_loss: 1.184
h 77/100
3 [==============================] - 0s 5ms/step - loss: 0.2597 - accuracy: 0.8910 - val_loss: 1.199
h 78/100
3 [==============================] - 0s 5ms/step - loss: 0.2591 - accuracy: 0.8923 - val_loss: 1.208
h 79/100
3 [==============================] - 0s 5ms/step - loss: 0.2515 - accuracy: 0.8950 - val_loss: 1.214
h 80/100
3 [==============================] - 0s 6ms/step - loss: 0.2465 - accuracy: 0.8976 - val_loss: 1.233
h 81/100
3 [==============================] - 0s 7ms/step - loss: 0.2404 - accuracy: 0.9010 - val_loss: 1.258
h 82/100
3 [==============================] - 0s 7ms/step - loss: 0.2376 - accuracy: 0.9019 - val_loss: 1.280
h 83/100
3 [==============================] - 0s 7ms/step - loss: 0.2357 - accuracy: 0.9027 - val_loss: 1.269
h 84/100
3 [==============================] - 0s 7ms/step - loss: 0.2347 - accuracy: 0.9028 - val_loss: 1.26
h 85/100
3 [==============================] - 0s 7ms/step - loss: 0.2300 - accuracy: 0.9063 - val_loss: 1.301
h 86/100
3 [==============================] - 0s 6ms/step - loss: 0.2262 - accuracy: 0.9078 - val_loss: 1.282
h 87/100
3 [==============================] - 0s 6ms/step - loss: 0.2221 - accuracy: 0.9098 - val_loss: 1.309
h 88/100
3 [==============================] - 0s 6ms/step - loss: 0.2187 - accuracy: 0.9107 - val_loss: 1.313
h 89/100
3 [==============================] - 0s 5ms/step - loss: 0.2139 - accuracy: 0.9136 - val_loss: 1.327
h 90/100
3 [==============================] - 0s 5ms/step - loss: 0.2091 - accuracy: 0.9140 - val_loss: 1.358
h 91/100
3 [==============================] - 0s 5ms/step - loss: 0.2093 - accuracy: 0.9151 - val_loss: 1.343
h 92/100
3 [==============================] - 0s 5ms/step - loss: 0.2075 - accuracy: 0.9168 - val_loss: 1.336
h 93/100
3 [==============================] - 0s 5ms/step - loss: 0.2037 - accuracy: 0.9176 - val_loss: 1.383
h 94/100
3 [==============================] - 0s 5ms/step - loss: 0.1992 - accuracy: 0.9207 - val_loss: 1.401
h 95/100
3 [==============================] - 0s 5ms/step - loss: 0.2008 - accuracy: 0.9206 - val_loss: 1.374
h 96/100
3 [==============================] - 0s 5ms/step - loss: 0.1975 - accuracy: 0.9203 - val_loss: 1.404
h 97/100
3 [==============================] - 0s 5ms/step - loss: 0.1940 - accuracy: 0.9235 - val_loss: 1.401
h 98/100
3 [==============================] - 0s 5ms/step - loss: 0.1919 - accuracy: 0.9240 - val_loss: 1.40
h 99/100
3 [==============================] - 0s 5ms/step - loss: 0.1888 - accuracy: 0.9254 - val_loss: 1.417
h 100/100
3 [==============================] - 0s 5ms/step - loss: 0.1848 - accuracy: 0.9278 - val_loss: 1.418
```

EVALUACIÓN Y METRICAS

```
# Guardamos los datos del entrenamiento
history_dict = history.history
history_dict
```

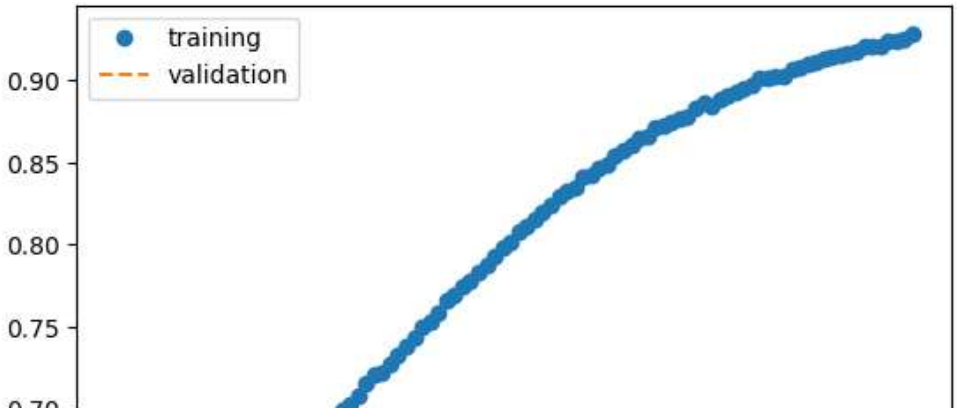
```
# Guardamos los valores de perdida en dos arreglos para entrenamiento y validacion
i=0
loss_values = history_dict["loss"][i:]
val_loss_values = history_dict["val_loss"][i:]

# Graficamos los valores de perdida
epoch = range(1, len(loss_values) + 1)
plt.plot(epoch, loss_values, 'o', label = 'training')
plt.plot(epoch, val_loss_values, '--', label = 'validation')
plt.legend()
plt.show()
```




```
# Guardamos los valores de precisión en dos arreglos para entrenamiento y validacion
i=0
loss_values = history_dict["accuracy"][i:]
val_loss_values = history_dict["val_accuracy"][i:]

# Graficamos los valores de precisión
epoch = range(1, len(loss_values) + 1)
plt.plot(epoch, loss_values, 'o', label = 'training')
plt.plot(epoch, val_loss_values, '--', label = 'validation')
plt.legend()
plt.show()
```




```
# evaluamos el modelo
model.evaluate(X_test, y_test)
```

```
625/625 [==============================] - 2s 2ms/step - loss: 1.4347 - accuracy: 0.6849
[1.4346693754196167, 0.6849499940872192]
```

```
# Predecimos en base a los datos de prueba
y_predict_prob = model.predict(X_test)

# Convertimos las probabilidades en clases
y_predict = np.argmax(y_predict_prob, axis=1)
```

```
625/625 [==============================] - 1s 2ms/step
```

```
y_predict_prob
```

```
array([[9.4015235e-01, 5.8574248e-02, 1.2733884e-03],
       [2.8551903e-01, 7.0787954e-01, 6.6014226e-03],
       [9.5051593e-01, 4.9483873e-02, 1.8077527e-07],
       ...,
       [1.4084031e-01, 8.0928218e-01, 4.9877524e-02],
       [7.8382927e-01, 2.1617073e-01, 3.4380732e-12],
       [2.4227962e-01, 6.0271271e-02, 6.9744909e-01]], dtype=float32)
```

```
# Verificamos algunos casos individuales
pos = 31
print("y: " , y_test[pos])
print("y predict:", y_predict[pos])
```

```
y:  1.0
y predict: 1
```

```
# Matriz de Confusion
print("Accuracy: {:.2f}%".format(accuracy_score(y_test, y_predict) * 100))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_predict))
```

```
Accuracy: 68.49%
Confusion Matrix:
[[6067 2864  194]
 [2526 7226  261]
 [ 224  232  406]]
```

Productos de pago de Colab - Cancelar contratos

✓ 0 s completado a las 13:41