

FACULTAD DE INGENIERÍA

Alumno: González Monsiváis Gerardo de Jesús, CU: 288385

Materia: Fundamentos de Inteligencia Artificial (12-13 PM)

Carrera: Ingeniería En Computación

Semestre 2023-2024/I

Profesor: Dr. Cuevas Tello Juan Carlos

Fecha: 04 de Diciembre de 2023

Resumen:

El presente proyecto presenta un clasificador de géneros musicales utilizando técnicas de aprendizaje automático y análisis de datos para poder determinar el género al que pertenece una melodía, en este caso se abordara solamente entre los géneros “Rock” y “Hip Hop” esto en base a los data set que nos proporciona la compañía echonest y que gracias a diversas métricas que nos proporcionan sus conjuntos de datos, podemos llegar a categorizar canciones en base a diversos atributos.

Una aplicación ya en la vida real de este tipo de clasificadores es en los servicios de música online como Spotify, deezer entre otros, los cuales buscan la forma de categorizar la música para así poder recomendarla a los usuarios en base a sus perfiles, esto implica un grande análisis de cada canción y para resolver este problema se usan estos clasificadores que sin procesar como tal el audio, se usan otros atributos de la canción para así determinar por medio de esas métricas el género al que pertenece.

Mas adelante en el desarrollo de este reporte se va a ir explicando paso a paso todo el proceso que se uso para poder construir este clasificador y así obtener las clasificaciones de las canciones en base a sus atributos propios.

Desarrollo:

Para entrar un poco mas a detalle de lo que vamos a usar en este proyecto para la creación del clasificador musical pues es formular un algoritmo que nos clasifique y nos prediga con una buena exactitud el tipo de género al que pertenece la canción, aquí usaremos un algoritmo de aprendizaje automático el cual nos va a generar una puntuación y la mejor puntuación será el género al que pertenece, para esto usaremos un árbol de decisión.

Para la parte del conjunto de datos que vamos a usar, será un data set que nos provee “Echo Nest” una empresa que pertenece a Spotify.

El conjunto de datos que usaremos este compuesto por canciones de dos tipos de géneros, Rock y Hip Hop esto para que sea más sencillo la clasificación por temas de tiempo y dicho dataset que se va a utilizar para el entrenamiento de este proyecto va a ser un dataset conocido como "echo-nest, la cual es una recopilacion de datos musicales pensada para desarrolladores. Este dataset está en formato json y basicamente una canción es más que su título, artista y número de escuchas" Otro de los set de datos que tenemos es uno de "Datacamp" el cual tiene características musicales de cada pista, como la capacidad de baile y la acústica en una escala de -1 a 1.

Implementación del código:

```
#Librerias a usar
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#Creamos una variable y leemos los datos que
están dentro de nuestro dataset, es decir las
pistas con etiquetas de género
tracks = pd.read_csv('fma-rock-vs-hiphop.csv')

#Creamos una variable y leemos las métricas de
seguimiento con las funciones
echonest_metrics = pd.read_json('echonest-
metrics.json', precise_float=True)

#Muestra más información de nuestra dataset
print(echonest_metrics.describe())
```

	track_id	acousticness	danceability	energy	\
count	13129.000000	1.312900e+04	13129.000000	13129.000000	
mean	34031.058268	5.246876e-01	0.487290	0.537516	
std	28950.422182	3.837186e-01	0.190148	0.278049	
min	2.000000	9.035000e-07	0.051307	0.000020	
25%	12986.000000	1.037726e-01	0.344759	0.321300	
50%	28097.000000	5.739848e-01	0.485635	0.549113	
75%	45021.000000	9.207270e-01	0.629094	0.776254	
max	124911.000000	9.957965e-01	0.968645	0.999964	

	instrumentalness	liveness	speechiness	tempo	\
count	13129.000000	13129.000000	13129.000000	13129.000000	
mean	0.640536	0.187804	0.099174	123.080061	
std	0.361430	0.158051	0.137381	35.015137	
min	0.000000	0.025297	0.022324	12.753000	
25%	0.323466	0.101406	0.036932	95.967000	
50%	0.838134	0.119002	0.049019	120.057000	
75%	0.918244	0.211041	0.085452	145.318000	
max	0.998016	0.980330	0.966177	251.072000	

	valence
count	13129.000000
mean	0.439761
std	0.276028
min	0.000010

```
#Hasta este punto ya tenemos como tal toda la
informacion de las canciones
#pero estan separadas en diferentes archivos, por
lo cual las vamos a ligar
#de acuerdo a su id

#Usamos pandas para hacer la union de la
informacion
echo_tracks = pd.merge(echonest_metrics,
tracks[["track_id", "genre_top"]], on='track_id')

# Observamos la informacion resultante de la
union
echo_tracks.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4802 entries, 0 to 4801
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  -
0   track_id              4802 non-null   int64
1   acousticness          4802 non-null   float64
2   danceability          4802 non-null   float64
3   energy                4802 non-null   float64
4   instrumentalness       4802 non-null   float64
5   liveness              4802 non-null   float64
6   speechiness           4802 non-null   float64
7   tempo                 4802 non-null   float64
8   valence                4802 non-null   float64
9   genre_top             4802 non-null   object
dtypes: float64(8), int64(1), object(1)
memory usage: 412.7+ KB

```

Para evitar duplicados (Redundancia) de funciones hacemos una matriz de correlación.

Matriz de correlación: Es una tabla que muestra los coeficientes de correlación entre todas las variables en un conjunto de datos.

El coeficiente de correlación de Pearson puede variar en un rango de -1 a 1, donde:

- * 1 indica una correlación positiva perfecta (ambas variables aumentan juntas en una relación lineal).
- * 0 indica ausencia de correlación.
- * -1 indica una correlación negativa perfecta (una variable disminuye mientras la otra aumenta en una relación lineal)

```
#Se crea una matriz de correlacion, por el
metodo pearson
corr_metrics = echo_tracks.corr(method='pearson')
#Dar estilo al mapa de calor
corr_metrics.style.background_gradient()
```

	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
track_id	1.000000	-0.372282	0.049454	0.140703	-0.275623	0.048231	-0.026995	-0.025392	0.010070
acousticness	-0.372282	1.000000	-0.028954	-0.281619	0.194780	-0.019991	0.072204	-0.026310	-0.013841
danceability	0.049454	-0.028954	1.000000	-0.242032	-0.255217	-0.106584	0.276206	-0.242089	0.473165
energy	0.140703	-0.281619	-0.242032	1.000000	0.028238	0.113331	-0.109983	0.195227	0.038603
instrumentalness	-0.275623	0.194780	-0.255217	0.028238	1.000000	-0.091022	-0.366762	0.022215	-0.219967
liveness	0.048231	-0.019991	-0.106584	0.113331	-0.091022	1.000000	0.041173	0.002732	-0.045093
speechiness	-0.026995	0.072204	0.276206	-0.109983	-0.366762	0.041173	1.000000	0.008241	0.149894
tempo	-0.025392	-0.026310	-0.242089	0.195227	0.022215	0.002732	0.008241	1.000000	0.052221
valence	0.010070	-0.013841	0.473165	0.038603	-0.219967	-0.045093	0.149894	0.052221	1.000000

Debido a la cantidad de datos que tenemos en nuestro data set usaremos un PCA para así poder reducir la cantidad de variables que no sean útiles para nuestro análisis.

```
# Define nuestras características para el
conjunto de datos y OJO aquí no vamos a usar
todas, ya que hay
# algunas que no nos van a interesar, entonces
con este drop vamos a eliminar el genero y el
track ID
features = echo_tracks.drop(['genre_top',
'track_id'], axis=1)

# Definimos esta etiqueta
labels = echo_tracks['genre_top']

# Import the StandardScaler
from sklearn.preprocessing import StandardScaler
```

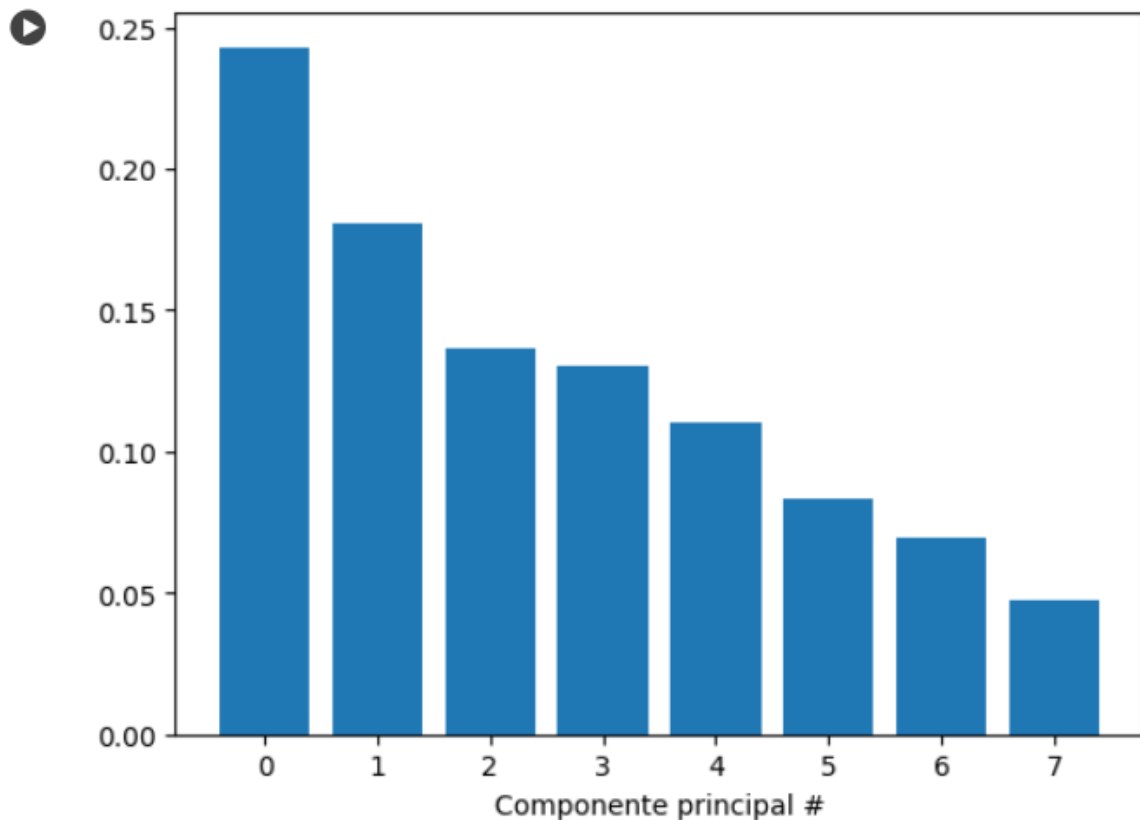
```
# Escalamos las características para que el
modelo pueda tomar una misma medición esto lo
hacemos
# cuando los datos no estan normalizados
scaler = StandardScaler()
scaled_train_features =
scaler.fit_transform(features)

# Con esta línea vamos a mostrar el grafico de la
varianza
%matplotlib inline

# Import our plotting module, and PCA class
from sklearn.decomposition import PCA

# Obtenemos las relaciones de la variación del PCA
pca = PCA()
pca.fit(scaled_train_features)
exp_variance = pca.explained_variance_ratio_

# Mostramos la varianza con un grafico de barras
fig, ax = plt.subplots()
ax.bar(range(8), exp_variance)
ax.set_xlabel('Componente principal #')
```



Podemos observar que este grafico no nos da mucha información que nos pueda ser útil para encontrar el numero de dimensiones, es muy limitada esa información por lo cual mejor vamos a crear un grafico de varianza acumulativa para así poder obtener cuantas características son las que se requieren.

```
# Calculo de la varianza acumulativa
```

```
cum_exp_variance = np.cumsum(exp_variance)
```

```
# Trazo de la varianza acumulativa y se hace un dibujo punteado en 0.90.
```

```
# Rango de 0 a 8 y de 0 a 10.
```

```
fig, ax = plt.subplots()
```

```
ax.plot(range(8), cum_exp_variance)
```

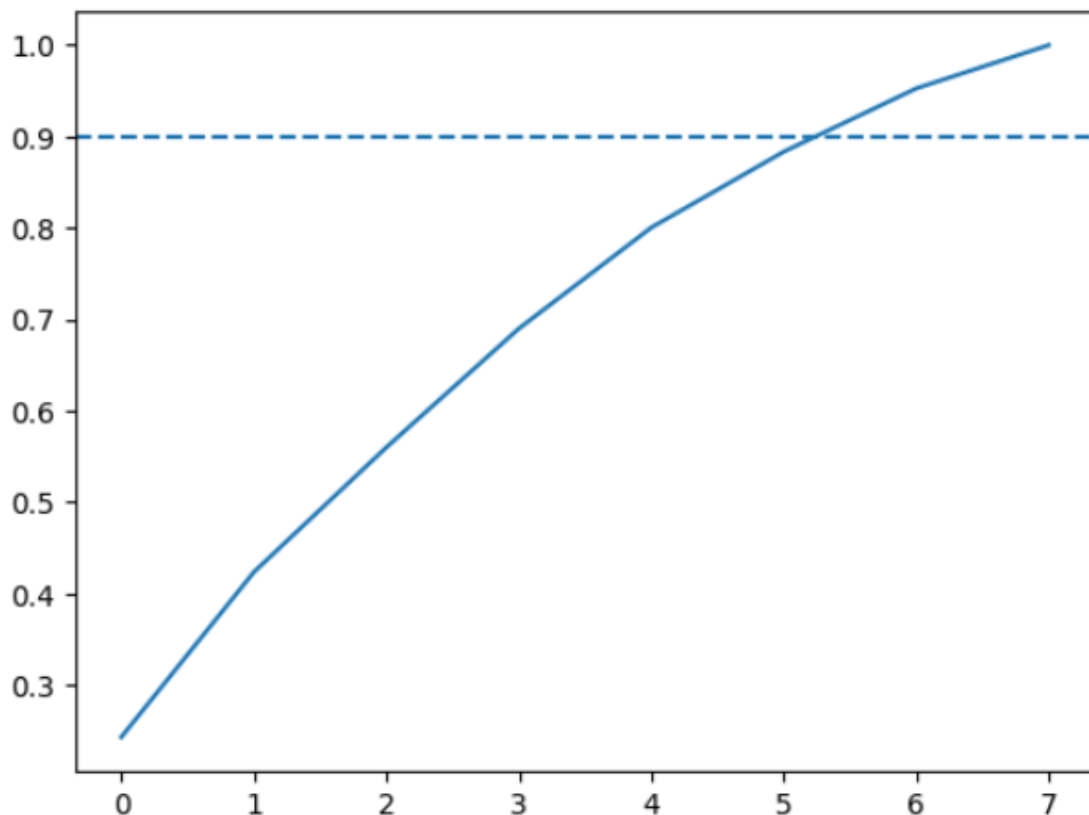


```

ax.axhline(y=0.9, linestyle='--')
n_components = 6

# Realizar PCA con el número elegido de
componentes y proyectar datos en componentes
pca = PCA(n_components, random_state=10)
pca.fit(scaled_train_features)
pca_projection =
pca.transform(scaled_train_features)

```



Árbol de Decisión:

Aquí vamos a crear un árbol de decisiones para así poder hacer la clasificación del género musical. Lo que vamos a hacer es dividir nuestro conjunto de datos en datos de entrenamiento y datos de prueba, en donde el conjunto de entrenamiento se usara para entrenar el modelo mientras

que el otro conjunto de prueba lo usaremos para evaluar y validar el rendimiento del modelo.

```
# Import train_test_split function and Decision
tree classifier
from sklearn.model_selection import
train_test_split
from sklearn.tree import DecisionTreeClassifier

# Dividimos nuestros para despues ocuparlos en
entrenamiento y prueba
train_features, test_features, train_labels,
test_labels = train_test_split(pca_projection,
labels, random_state=10)

# Entrenamos el arbol con los datos de
entrenamiento
tree = DecisionTreeClassifier(random_state=10)
tree.fit(train_features, train_labels)

# Predecimos las etiquetas resultantes con los
datos de prueba
pred_labels_tree = tree.predict(test_features)
```

Obtenemos nuestro árbol ya clasificado y entrenado

Aquí vamos a obtener lo siguiente:

- Precisión: indica que porcentaje de las predicciones fueron correctas
- Recall: Capacidad del clasificador para encontrar las instancias positivas, es decir falsos positivos y falsos negativos.
- Puntuación F1: puntuación o score del modelo
- Apoyo: numero de apariciones reales de la clase en el conjunto de datos especificado.

Obtención del reporte de clasificación del modelo ya entrenado:

```
# Creamos la clasificacion del reporte de nuestro
modelo
from sklearn.metrics import classification_report
class_rep_tree =
classification_report(test_labels,
pred_labels_tree)
print("Decision Tree: \n", class_rep_tree)
```

```
Decision Tree:
              precision    recall  f1-score   support

   Hip-Hop       0.60      0.60      0.60        235
    Rock       0.90      0.90      0.90        966

 accuracy              0.84        1201
 macro avg       0.75      0.75      0.75        1201
 weighted avg    0.84      0.84      0.84        1201
```

En esta parte ya podemos observar que la precisión para las canciones de Hip Hop esta algo baja mientras que para las canciones de Rock esta algo elevada, en el recall podemos observar que en Hip Hop esta en el 0.60% y esto quiere decir que esa fue la capacidad que tuvo el modelo para encontrar falsos positivos y de igual forma sigue siendo algo baja para el caso del hip hop y en el caso de soporte que es el numero de registros con los que se esta trabajando, vemos que de igual forma esta muy bajo y es por ello que vamos a equilibrar esos datos para que no haya un desbalanceo entre ambos géneros y haya una mejor precisión.

Observando nuestro reporte podemos ver que las canciones de rock están bastante bien clasificadas, pero las canciones de hip hop no, están desproporcionalmente mal clasificadas como canciones de rock y esto se debe a que tenemos muchos mas puntos de datos para la clasificación de Rock que para la de Hip Hop y esto hace que haya un sesgo en el modelo

y la capacidad para clasificar y distinguir las dos clases y esto hace una mayor precisión para la clasificación del género Rock.

Para resolver esto vamos a re dimensionar el set de datos.

```
# Hacemos un subconjunto de datos, solo vamos a
tomar los datos de echo track con su campo de
genero tanto para hip hop y rock
hop_only =
echo_tracks.loc[echo_tracks['genre_top'] == 'Hip-
Hop']
rock_only =
echo_tracks.loc[echo_tracks['genre_top'] ==
'Rock']

# muestrea las canciones de rock para que sean el
mismo número de las canciones que hay de hip-hop
rock_only = rock_only.sample(len(hop_only),
random_state=10)

# Concatenamos los datos  datos rock_only y
hop_only
rock_hop_bal = pd.concat([rock_only, hop_only])

# Con las características de las etiquetas y la
proyección del PCA vamos a crear el marco de
datos para equilibrarlo
# y se quite el sesgo
features = rock_hop_bal.drop(['genre_top',
'track_id'], axis=1)
labels = rock_hop_bal['genre_top']
pca_projection =
pca.fit_transform scaler.fit_transform(features))
```


```
# Redefinimos el conjunto de datos para entrenar
y volvemos a usar los datos para que esten
balanceado
train_features, test_features, train_labels,
test_labels = train_test_split(pca_projection,
labels, random_state=10)
```

Análisis del segundo reporte ya con el balanceo de datos:

Aquí ya hicimos el equilibrio con los datos nuevos y vamos a ver si en realidad ese equilibrio nuevo mejoro el sesgo que había.

```
# Entrenamiento del nuevo arbol ya con los datos
balanceados
tree = DecisionTreeClassifier(random_state=10)
tree.fit(train_features, train_labels)
pred_labels_tree = tree.predict(test_features)

# Mostramos el reporte del nuevo arbol
print("Decision Tree: \n",
classification_report(test_labels,
pred_labels_tree))
```

 Decision Tree:				
	precision	recall	f1-score	support
Hip-Hop	0.74	0.73	0.74	230
Rock	0.73	0.74	0.73	225
accuracy			0.74	455
macro avg	0.74	0.74	0.74	455
weighted avg	0.74	0.74	0.74	455

Conclusiones:

Como conclusión podemos ver que este modelo que se uso para clasificar canciones trabaja de forma correcta es decir tiene una efectividad por arriba del 70 % para clasificar canciones de acuerdo a su género, que para fines de este proyecto se limitó solo a dos géneros en particular. Además durante la realización de este proyecto me pude dar cuenta de que es de suma importancia elegir bien nuestros data set además de que aunque tengamos un conjunto de datos muy grande no nos podemos confiar ya que como se pudo observar tuvimos que hacer una re dimensionar ese conjunto de datos ya que esto causaba un sesgo en la clasificación de los géneros y esto nos trajo como consecuencia que las canciones del genero rock las clasificara de forma eficiente pero para las canciones de Hip Hop las clasificaba de forma errónea en un gran porcentaje y esto fue a que había mas canciones de Rock que de Hip Hop y esto hacia ese sesgo, esto se pudo resolver haciendo una re dimensión de estos datos y pues al final pudimos observar que haciendo esos cambios necesarios ya nos dio un puntaje mayor para nuestras métricas las cuales eran la precisión, el recall, el score y el support. Con esto concluimos que de acuerdo con las entradas que le damos tanto en el conjunto de prueba y de entrenamiento, puede afectar el rendimiento del modelo.

Para finalizar esta opinión es un poco mas personal y siento que el desarrollo de este proyecto me ayudo a reforzar los conocimientos obtenidos en clase ya que aquí se tocaron algunos de los temas que estuvimos checando durante todo el semestre como lo son los árboles de decisión, el lenguaje de programación Python y algunos otros temas más.

Bibliografia:

https://es.wikipedia.org/wiki/The_Echo_Nest

<https://machinelearningparatodos.com/arboles-de-decision-en-python/>

<https://www.kaggle.com/datasets/veronikafilippou/echonestmetricsjson/>

<https://www.delftstack.com/es/howto/python-pandas/pandas-correlation-matrix/>