Fundamentos del Procesamiento de Lenguaje Natural (NLP)

Clasificación de canciones por generos

Iker Ruesgas Escalante
Alejandro Contreras Alegria

ÍNDICE:

1. Propuesta de proyecto - E1

1.1 Objetivo

Nuestro proyecto tiene como objetivo el desarrollo de un modelo de procesamiento del lenguaje natural que permita a partir de la letra de una canción identificar el género de una canción. Para ello, utilizaremos un dataset que incluye las letras de las canciones, junto con etiquetas correspondientes de género o emociones asociadas a los géneros que tratan dichas canciones. Tenemos ya un dataset seleccionado, sin embargo, tenemos otros dos por si cambiamos el enfoque o es necesario utilizar otro.

El enfoque principal que tenemos pensado implica aplicar diversas técnicas de NLP para analizar las letras de las canciones. Entre estas tareas que tenemos pensadas están incluidas la tokenización, la limpieza de texto (eliminación de stopwords, normalización) y la extracción de características utilizando representaciones como TF-IDF o embeddings. Si en un futuro optamos por el análisis de sentimientos, emplearemos modelos de clasificación supervisada para asignar emociones como alegría, tristeza o enfado. Por otro lado, si nos enfocamos en la predicción del género musical, utilizaremos modelos supervisados para identificar géneros como rock, pop, jazz, rap, entre otros.

El modelo se entrenará y evaluará utilizando métricas de rendimiento como precisión, recall y F1-score (si vemos que no nos generan los resultados previstos las cambiaremos por otras). Para evitar el sobreajuste y mejorar la robustez del modelo, aplicaremos validación cruzada, dividiendo el dataset en varias partes para entrenarlo evitando así el overfitting. Además, exploraremos diferentes arquitecturas de modelos, incluyendo redes neuronales recurrentes (RNN, ya que lo hemos visto antes en Deep Learning) y transformers, con el objetivo de optimizar los resultados.

1.2 Datasets

1.2.1 Dataset por géneros principal:

- Link:
  https://www.kaggle.com/datasets/carlosgdcj/genius-song-lyrics-with-language-information
- Descripción:
  Este dataset contiene una extensa colección de letras de canciones de la plataforma Genius. En este conjunto de datos no solo se incluyen letras, sino también

información extra como el idioma en el que están escritas, de esta manera podemos filtrar solo las que están en inglés.

- Tamaño:

```
tamaño = dt.shape
print(f"El dataset tiene {tamaño[0]} filas y {tamaño[1]} columnas.")
✓ 0.0s
El dataset tiene 5134856 filas y 11 columnas.
```

Feature	Descripción	Tipo	Característica
title	Título	String	3 millones de valores únicos
tag	Género	String	pop 42% y rap 34%
artist	Autor	String	640 mil valores únicos
year	Año de publicación	Integer	
views	Número de visitas	Integer	
features	Información adicional	String	77% de esta feature está vacía
lyrics	Letras	String	5 millones de valores únicos
id	id	Integer	
language_cld3	Idioma cld3	String	
language_ft	Idioma fast text	String	
language	Idioma	String	Comprueba si cld3 y fast text son iguales

- Anotaciones:
  - Tamaño útil del dataset:
    Como este dataset contiene canciones en distintos idiomas, y solo nos interesan las canciones en inglés, vamos a filtrar directamente las que no estén en este idioma. De esta manera sabremos el tamaño real del que partiremos antes de hacer el data cleaning y data preprocessing.

```
tamaño = dt.shape
print(f"El dataset tiene {tamaño[0]} filas y {tamaño[1]} columnas.")
✓ 0.3s
El dataset tiene 3374198 filas y 11 columnas.
```

  - Contenido:

```
title          127
tag              0
artist           0
year             0
views            0
features         0
lyrics           0
```

Para ver de manera general el contenido de las columnas vamos a ver si contienen muchos valores nulos.
Podemos comprobar que solo unas pocas canciones tienen valores nulos en su título, pero en general nada relevante. Aunque no aparezcan valores nulos en la columna *features* el 77% de los arrays están vacíos.

- Uso del dataset:
  Este dataset será el principal y en el que nuestro proyecto girará en torno. Contiene las features necesarias para cumplir nuestro objetivo además de ser extenso. En caso de que este no nos sea útil o queramos dar otro enfoque tenemos otros dos datasets.

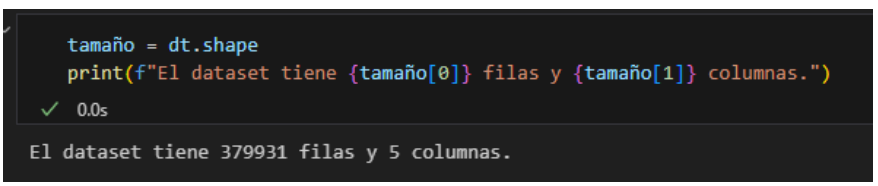
1.2.2 Dataset por géneros secundario:

- Link:
  https://www.kaggle.com/datasets/neisse/scrapped-lyrics-from-6-genres
- Descripción:
  Este dataset está dividido en dos csv. El primero contiene información sobre el artista mientras que el segundo contiene las canciones con sus respectivas letras.

1. ***artists-data.csv***
   - Tamaño:
     - Columnas: 5
     - Filas: 4169

Feature	Descripción	Tipo	Característica
Artist	Artista	String	4168 valores únicos
Genres	Género	String	Array de géneros
Songs	Número de canciones	String	
Popularity	Puntuación de popularidad	String	
Link	Link	String	4168 valores únicos

2. ***lyrics-data.csv***
   - Tamaño:



Feature	Descripción	Tipo	Característica

ALink	Link	String	4239 valores únicos
SName	Nombre de la canción	String	267259 valores únicos
SLink	Link a la canción	String	379893 valores únicos
Lyric	Letra	String	
language	Idioma	String	50% en ingles y 41% en portugues

- Anotaciones:
  - Estructura:
    Este dataset está estructurado en forma de tablas anidadas. El primer csv, artists-data.csv, tiene información sobre los artistas y lyrics-data.csv contiene las canciones de esos artistas. Por cada artista del primer csv hay una o varias canciones en lyrics-data.csv.
  - Tamaño útil del dataset:
    Al igual que con el dataset 1 vamos a eliminar directamente todas las canciones que no estén en inglés, ya que no nos interesan para nuestro trabajo, así sabremos realmente de cuantas filas disponemos.

```
tamaño = dt.shape
print(f"El dataset tiene {tamaño[0]} filas y {tamaño[1]} columnas.")
✓ 0.0s
El dataset tiene 191814 filas y 5 columnas.
```

  - Contenido:
    En este dataset tampoco hay ninguna columna con muchos valores nulos así que tampoco hay ningún problema.

```
ALink         0
SName         2
SLink         0
Lyric         0
language      0
dtype: int64
Artist        1
Genres        5
Songs         1
Popularity    2
Link          1
dtype: int64
```

  - Uso del dataset:
    En principio este dataset no lo usaremos y el principal será el anterior. Sin embargo, si el primer dataset no nos es útil, nos da problemas por su extensión o simplemente queremos añadir otro apartado al proyecto contamos con este dataset como reserva.

1.2.3 Dataset para emociones:

- Link: https://www.kaggle.com/datasets/saurabhshahane/music-dataset-1950-to-2019
- Descripción:
  Este dataset contiene información sobre las canciones y sus letras. Además de incluir el género, lo que diferencia este dataset de los anteriores es que se incluyen varias features que representan emociones o sentimientos en un valor de 0 a 1.
- Tamaño:

Columnas: 31
Filas: 28273

Feature	Descripción	Tipo	Característica
artist_name	Artista	String	5426 valores únicos
track_name	Nombre de la canción	String	23689 valores únicos
release_date	Fecha de salida	Integer	1950 a 2019
genre	Género	String	
lyrics	Letra	String	
len	Número de palabras	Integer	
dating		Decimal	
violence		Decimal	
world/life		Decimal	
night/time		Decimal	
shake the audience		Decimal	
family/gospel		Decimal	
romantic		Decimal	
communication		Decimal	
obscene		Decimal	
music		Decimal	
movement/places		Decimal	
light/visual perceptions		Decimal	
family/spiritual		Decimal	
like/girls		Decimal	
sadness		Decimal	
feelings		Decimal	
danceability		Decimal	

loudness		Decimal	
acousticness		Decimal	
instrumentalness		Decimal	
valence		Decimal	
energy		Decimal	
topic	Tema principal	String	
age		Decimal	

- Anotaciones:
  Uso del dataset:
  A diferencia de los anteriores datasets este contiene información sobre los sentimientos que transmite la canción en una escala normalizada. Este dataset nos será útil si en vez de clasificar las canciones por géneros queremos hacerlo por sentimientos.

1.3 Análisis del estado del arte

- **Proyecto "NLP_Song_Lyrics_Based_Recommendation":**
  - Link:
    https://github.com/rk150888/NLP_Song_Lyrics_Based_Recommendation
  - Descripción:
    Este proyecto trata sobre un sistema de recomendación de canciones basado en sentimientos. Las letras de la canción están clasificadas en positivas, negativas o no definidas. El training data sigue estos pasos:
    1. Función para identificar letras y sentimientos a partir de datos de entrenamiento.
    2. Eliminar las Stopwords.
    3. Crear la bagwords de las letras.
    4. Determinar la distribución de frecuencia de las letras.
    5. Aplicar features al corpus filtrado y aplicar Naive-Bayes para entrenar el clasificador en los datos de entrenamiento.

    Por último, para el testing:
    1. Aplicar el clasificador en el testing data para identificar los sentimientos en las letras.
    2. Calcular la accuracy del modelo.

    Este proyecto tiene un enfoque muy similar a lo que queremos hacer en nuestro trabajo por lo que nos puede ser útil.

- **Recomendador de Spotify:**
  - Link: https://support.spotify.com/ni/article/your-taste-profile/
  - Descripción:

    Aparte de ejemplos más concretos como el proyecto que hemos encontrado en gitHub, también hay otras aplicaciones más cotidianas que usamos en el dia a dia como Spotify, está entre otros aspectos destaca por la capacidad de analizar las preferencias musicales de los usuarios. Utiliza algoritmos que examinan las canciones que hemos escuchado previamente, creando de esta manera listas de reproducción personalizadas. Esta aplicación saca partido de tener una de las mayores bases de datos relacionados con letras de canciones y usuarios para utilizarlo como recomendador de canciones

    No está detallado exactamente qué algoritmos o técnicas utilizan pero sin duda entre ellas habrá técnicas de NLP, por ejemplo, Spotify ofrece listas de reproducción basándose en emociones. Para ello al tener gran mayoría de sus canciones concatenadas con sus letras pueden aplicar un sistema de reconocimiento de sentimientos para ir actualizando sus listas.

1.4 Resumen

En cuanto a esta primera entrega, prácticamente hemos logrado todo lo que nos proponemos. En primer lugar, hemos utilizado las distintas técnicas de procesamiento como tokenizar, limpiar o el uso de TF-IDF como habíamos planteado. También entre las dos iteraciones del trabajo hemos conseguido probar distintos modelos y enfoques como habíamos pensado. En cuanto a la elección del dataset, la primera propuesta que hicimos ha sido la que finalmente hemos estado utilizando a lo largo del trabajo. En principio teníamos en mente que cuantas más filas y columnas mejor vendría aunque ha medida que hemos ido avanzando nos hemos dado cuenta que trabajar con uno tan grande era complicado con los tiempos de entrenamiento de modelos.

2. Procesamiento y análisis de los datos - E2

2.1 Introducción

2.1.1 Contexto

Como explicamos en el primer análisis, este trabajo tiene como objetivo el desarrollo de un modelo de procesamiento del lenguaje natural que permita a partir de la letra de una canción predecir su género musical. Este análisis se enfocará en procesar las letras de las canciones y representarlas con distintos métodos para conocer más a fondo nuestro dataset.

2.1.2 Dataset

En cuanto al dataset, escogimos nuestra primera opción que planteamos la última vez. Sin embargo, como ya dijimos anteriormente este dataset es muy amplio y nos hemos encontrado con que al hora de hacer los procesos tardaba demasiado o que incluso no podíamos llegar a cargar el dataset. Lo que hemos decidido ha sido coger una muestra y trabajar con ella más cómodamente. Posteriormente, con el trabajo finalizado intentaremos hacer una ejecución final con el máximo número de canciones

Para nuestra muestra del dataset hemos eliminado las columnas 'features', 'language_ft' y 'language_cld3' ya que o bien estaban vacías o no aportan nada para nuestro propósito. Por otro lado, hemos calculado cuántas canciones hay por cada género, siendo pop el género más popular y además podemos ver que hay un total de 6 géneros distintos.

2.2 Tokenización

Para nuestro análisis vamos a necesitar tokenizar las canciones por palabras y por oraciones. En primer lugar, lo que vamos a hacer es limpiar las canciones para que se puedan separar correctamente por tokens. Al ver una letra de una canción completa lo que podemos encontrar es qué hay comentarios que están delimitados por corchetes. En estos comentarios lo que se indica es en qué verso se encuentra, si es la intro, el final… básicamente información de la estructura. Estos comentarios los vamos a eliminar ya que no aportan ninguna información sobre el contexto y no es un indicador de que pertenezca a un género u otro. Por otro lado vamos a eliminar todos los caracteres que no sean números o letras y por último quitamos los espacios. Esta función se la aplicamos a todo el dataset e imprimimos una letra para comprobar que quede el resultado que queríamos. Por último, hemos tenido bastantes problemas con el tema de las contracciones ya que no sabíamos si dejar de o no la comilla. Cuando la dejábamos la tokenización no se hacía correctamente y separaba palabras que no eran y cuando la eliminamos algunos stopword no se reconocían correctamente. En un principio los mejores resultados nos daban eliminando la comilla pero

finalmente hemos encontrado la librería de “contractions” que nos ha solucionado este problema ya que nos permite dividir en las dos palabras originales.

Como ya tenemos las canciones en el formato que buscábamos, dividimos el dataset en train, validiation y test. A partir de aquí los análisis y comprobaciones que hagamos se harán con el train data. Además, ahora ya podemos empezar con ambas tokenizaciones. Para la función que divide por oraciones la mejor manera que hemos visto de dividir era cuando hubiese un salto de línea ya que las canciones no usan signos de puntuación como el punto o la coma para separar las frases. Por otro lado, para la función que divide por palabras simplemente hemos usado el TreebankWord tokenizer y eliminado las stopwords, imprimimos algunas filas y confirmamos que el resultado es lo que buscábamos. Por último, creamos el bag of words e imprimimos las palabras que más se repiten.

2.3 TF-IDF

Para representar la importancia de cada palabra en las letras de las canciones, hemos utilizado la técnica TF-IDF . Esta medida nos permite identificar qué términos son importantes dentro de un conjunto de texto.

El Term Frequency es una medida que indica la cantidad de veces que aparece una palabra en un texto. Sin embargo, palabras como "the" o "and" suelen aparecer mucho pero no aportan información única. Por eso, el TF por sí solo no es suficiente para sacar el peso de las palabras importantes. Una de las primeras observaciones que hicimos fue que algunos términos, como “na” y “ba”, aparecen muy frecuentemente en el top de las palabras con mayor TF. Esto es lógico en letras de canciones donde estos sonidos se repiten en coros o estribillos. También encontramos palabras más específicas como “soul” y “love”, que son representativas de los géneros musicales analizados.

En el caso del Inverse Document Frequency, para evitar el ruido de palabras comunes, calculamos la frecuencia inversa de documentos, que disminuye el peso de palabras más comunes en nuestro dataset. Así, términos menos frecuentes en el corpus pero relevantes en una canción específica tendrán un valor TF-IDF más alto. Al imprimir las palabras con menos IDF encontramos “know” o “like” que son comunes en muchas canciones y, por lo tanto, tienen un valor bajo en IDF. Esto significa que, aunque estas palabras aparecen con alta frecuencia, su contribución a la singularidad de una canción en particular es baja. En cambio, palabras con un IDF más alto aparecen con mucha menos frecuencia en el corpus general, lo que les da un peso de importancia mayor en las canciones donde aparecen.

Para cada palabra, multiplicamos su TF e IDF, generando un valor de TF-IDF. Este número ayuda a resaltar los términos que son más importantes en cada letra de canción. Hemos generado las palabras con mayor y menor TF-IDF en el dataset, aquellas que tienen alto TF-IDF tienden a reflejar temas o términos únicos, mientras que las de bajo TF-IDF incluyen conectores y palabras muy repetitivas. Al ver las palabras con más y menos TF-IDF no vemos ningún patrón o algo de lo que podamos sacar conclusiones.

2.4 Embeddings

2.4.1 No contextuales: Word2Vec

Word2Vec es un modelo de aprendizaje que convierte palabras en vectores numéricos. En vez de aprender el significado de una palabra, predice cuál es la palabra más probable que aparezca cerca de otra. Las palabras que estén en contextos similares tendrán vectores cercanos entre sí.

Inicialmente, para nuestro trabajo, vamos a usar el modelo original de Mikolov el cual ya está entrenado y tiene su vocabulario. Si vemos que es necesario nos planteamos entrenar nuestro propio modelo. Primero, lo que hemos hecho en nuestro código es contar el número de palabras lo que nos da un total de 1081960. De ese número de palabras hemos comparado cuántas no están representadas en el Word2Vec siendo 62555 lo que supone 5.78% de nuestro corpus. En principio esto no supone un problema por lo que podríamos continuar, pero para asegurarnos vamos a ver una muestra de palabras que no estén representadas. Podemos ver que por lo general son palabras mal escritas, inventadas o nombre osea lo que esperábamos, sin embargo, si calculamos las palabras no representadas que más se repiten podemos ver que son todas de ese tipo por lo que podemos confirmar que los anteriores procesos como la tokenización o el mismo Word2Vec se han hecho correctamente. Por último hemos cogido las palabras que más se repetían del bag of words y las hemos analizado con el Word2Vec , el resultado es que reconoce todas esas palabras.

```
Total de palabras en el dataset: 1081960
Palabras no representables con Word2Vec: 62555
Porcentaje de palabras no representables: 5.78%
```

2.4.2 Contextuales: ELMo

A diferencia de Word2Vec, ELMo se basa en una arquitectura que genera embeddings a partir de los caracteres en lugar de depender de un vocabulario fijo. Lo que hace exactamente es que va procesando el texto carácter a carácter, de izquierda a derecha y de derecha a izquierda. Por lo tanto todas las palabras se pueden representar sin importar que no estén dentro de un vocabulario o no existan. En nuestro caso nos viene bien ya que en muchas canciones se suelen usar palabras que no están reconocidas o son neologismos.

En cuanto al código, hemos cargado el modelo de ELMo y realizado una prueba utilizando una muestra de 3 frases para asegurarnos de que funcione correctamente. Posteriormente mediante alguna red neuronal volveremos a usar elmo. El resultado que hemos obtenido a partir del modelo ELMo es un tensor que representa los embeddings de las palabras en el contexto de las canciones. Lo que nos devuelve es un array de características que podremos utilizar para entrenar nuestro modelo de clasificación de géneros.

2.5 Resumen

Para esta segunda entrega lo que hicimos fue hacer procesamiento del dataset y algunos análisis. Para empezar, tuvimos que limpiar el dataset ya que lo que decidimos que más nos interesaba eran las letras como tal y decidimos eliminar comentarios que se suele añadir en las canciones. En cuanto a la tokenización hicimos dos métodos el de por palabras y el de oraciones. Para el primero, nuestra mayor complicación fue el tema de contracciones que finalmente gracias a una librería que encontramos pudimos resolverlo y detectaba correctamente las palabras. En cuanto a la función de oraciones, decidimos que lo más lógico, debido al formato en el que teníamos las canciones, era que se dividiese cuando hay un salto de línea ya que en las canciones no hay signos de puntuación para dividir las frases. Para el apartado de TF-IDF lo que hicimos fue usar la versión de clase pero esta era una versión simplificada. Posteriormente, para la siguiente entrega sustituimos este método por el de Tfidfvectorizer como se nos dijo en la corrección. Por último, para el tema de los embedding hicimos uno por cada tipo. Para los no contextuales utilizamos el Word2Vec que nos dio unos buen resultado de casi el 95% por ciento de nuestro corpus estaba reconocido por lo que posteriormente en la siguiente entrega es por lo que usamos estos embedding. Por otro lado, para los contextuales se usó el modelo de ELMo.

3. Primera iteración de resolución del problema - E3

3.1 Introducción

3.1.1 Contexto

En esta tercera entrega, el trabajo tiene como objetivo clasificar el género de las canciones usando técnicas de NLP y aprendizaje automático. Para ello preprocesamos las letras de las canciones y entrenamos dos tipos de modelos diferentes (Shallow y Deep Learning) y por último evaluamos su desempeño.

3.1.2 Dataset

En esta entrega nos hemos dado cuenta de que el dataset tenía una distribución desequilibrada de géneros, con algunos géneros teniendo muchas más canciones que otros. Por lo tanto creemos que esto podría sesgar el modelo y hacer que predijera los géneros más frecuentes con mayor frecuencia, sin aprender realmente las características de los géneros menos frecuentes. Para solucionar esto hemos decidido reducir el número de canciones en los géneros más representados para que todos los géneros tengan un número similar de canciones. De esta forma se equilibra el dataset y creemos que se evitará que el modelo se sesgue.

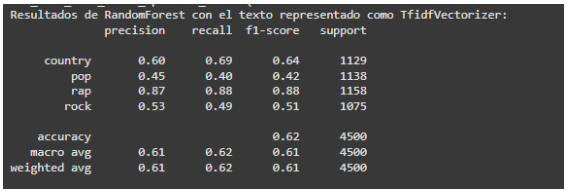
3.2 Clasificación con shallow machine learning

Modelo	Género	Precisión	Recall	f1-score	Support
Random Forest	country	0.60	0.69	0.64	1129
	pop	0.45	0.40	0.42	1138
	rap	0.87	0.88	0.88	1158
	rock	0.53	0.49	0.51	1075
Logistic Regression	country	0.65	0.69	0.67	1129
	pop	0.43	0.36	0.39	1138

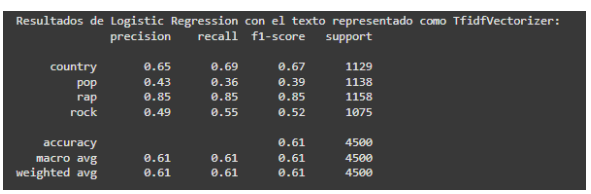
	rap	0.85	0.85	0.85	1158
	rock	0.49	0.55	0.52	1075
Decision Tree	country	0.47	0.48	0.47	1129
	pop	0.32	0.33	0.32	1138
	rap	0.77	0.73	0.75	1158
	rock	0.39	0.39	0.39	1075
Linear SVM	country	0.62	0.67	0.65	1129
	pop	0.43	0.38	0.40	1138
	rap	0.85	0.85	0.85	1158
	rock	0.47	0.49	0.48	1075
Non-linear SVM	country	0.66	0.67	0.66	1129
	pop	0.43	0.40	0.41	1138
	rap	0.87	0.83	0.85	1158
	rock	0.50	0.56	0.53	1075

Hemos entrenado 5 modelos de shallow learning (Random Forest, Logistic Regression, Decision Tree, Linear SVM y Non-linear SVM) usando TfidfVectorizer para representar las letras de las canciones como vectores numéricos.Como se puede observar en la tabla anterior hemos evaluado el rendimiento de cada modelo con métricas como accuracy, precision, recall y F1-score.

Viendo los datos de la tabla podemos observar que los modelos de shallow learning logran un buen rendimiento en clasificar canciones por género, con valores de accuracy y F1-score relativamente altos en algunos casos. Además de esto, Random Forest y Logistic Regression son los modelos que mejor rendimiento han obtenido con una accuracy del 61% y 62%.


Resultados de RandomForest con el texto representado como TfidfVectorizer:

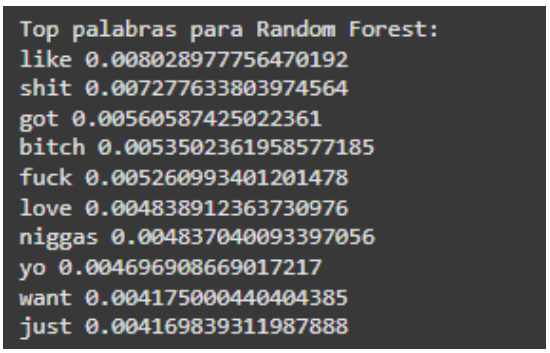
	precision	recall	f1-score	support
country	0.60	0.69	0.64	1129
pop	0.45	0.40	0.42	1138
rap	0.87	0.88	0.88	1158
rock	0.53	0.49	0.51	1075
accuracy			0.62	4500
macro avg	0.61	0.62	0.61	4500
weighted avg	0.61	0.62	0.61	4500

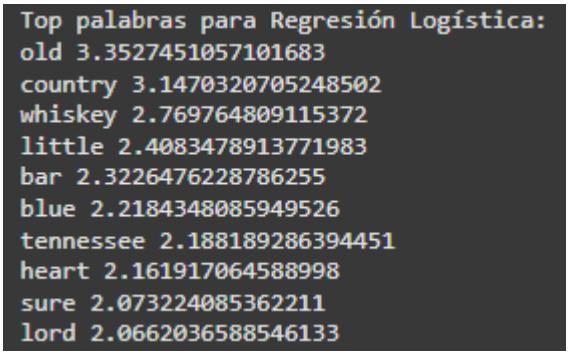

Resultados de Logistic Regression con el texto representado como TfidfVectorizer:

	precision	recall	f1-score	support
country	0.65	0.69	0.67	1129
pop	0.43	0.36	0.39	1138
rap	0.85	0.85	0.85	1158
rock	0.49	0.55	0.52	1075
accuracy			0.61	4500
macro avg	0.61	0.61	0.61	4500
weighted avg	0.61	0.61	0.61	4500

Por otra parte, hemos identificado las palabras más importantes para Random Forest y Logistic Regression. Estas palabras son las que más influyen en la

predicción del género y pueden dar información sobre las características de cada género musical.





3.3 Clasificación con CNN

Para esta parte del proyecto, nos enfocamos en dos configuraciones distintas utilizando arquitecturas de redes neuronales convolucionales. Además para cada arquitectura vamos a probar con embeddings pre entrenados y entrenando los embeddings. La arquitectura CNN_NLP es más sencilla y eficiente, con tres capas convolucionales y dropout. La Classifier es más profunda y compleja pero tiene un mayor riesgo de sobreajuste.

3.3.1 Classifier con y sin embeddings

Al trabajar con el clasificador CNN sin embeddings pre entrenados, el modelo necesita aprender desde cero las representaciones de las palabras. Esto significa que depende totalmente de nuestro dataset para entender el significado de las palabras y la relación que hay entre ellas. Como esperábamos, los primeros valores fueron más bajos y el modelo fue aprendiendo más lento. Sin embargo, con el paso de los epochs, alcanzó valores similares al modelo con embeddings, aunque con algo más de pérdida lo que sugiere que el modelo se sobreajusta.

Por otro lado, el clasificador con embeddings pre entrenados con valores más altos logró una convergencia más rápida. Eso tiene sentido porque los embeddings ya captaron las relaciones generales de las palabras, haciendo que el modelo se centre en patrones más específicos. Sin embargo, al no ajustar completamente los embeddings a las letras de las canciones, el modelo llega a un punto en el que prácticamente no varía dejándonos un valor máximo de accuracy del 58%. Sabemos que la opción con embeddings pre entrenados es la que mejor resultados dará, en nuestro caso nos faltaria fine tunear el modelo para conseguir mejores resultados pero el entorno nos está dando muchos problemas y sin el uso de cuda no podemos probar más configuraciones.

3.3.2 NLP con y sin embeddings

En el caso de esta arquitectura esperamos encontrar mejores resultados que con classifier, esto se debe a que esta arquitectura está diseñada específicamente para trabajar con texto la cual tiene en cuenta la posición de las palabras y la relación entre ellas lo cual teniendo en cuenta que nuestro dataset es de letras canciones debería ser mejor.

Teniendo desactivados los embeddings el modelo tuvo que aprender estas relaciones desde cero esto implica que el aprendizaje será más lento aunque al igual que la anterior arquitectura si que llega a valores prácticamente iguales que usando embeddings. En la configuración con embeddings pre entrenados, la red mostró un mejor desempeño al inicio porque ya partía de una representación semántica. Sin embargo, esta ventaja inicial no fue suficiente para obtener resultados significativamente mejores en los epochs finales.

Arquitectura	Embeddings	Epoch	Accuracy	Loss
Classifier	False	4	57.01	1.03
	True	4	58	0.98
NLP	False	2	63.02	0.85
	True	1	64.03	0.83

3.4 Clasificación con LSTM

Usando LSTM primero convertimos las palabras de las letras en vectores numéricos mediante embeddings(GloVe), que son como mapas matemáticos que muestran las relaciones entre las palabras. Luego, el modelo analiza estas secuencias y aprende patrones que lo ayudan a adivinar a qué género pertenece cada canción. Al combinar LSTM con GloVe, buscamos capturar tanto la estructura de las frases como los matices de significado entre palabras.

Los resultados obtenidos nos muestran que esta tarea no es tan sencilla. Puede que el modelo no sea capaz de distinguir bien los géneros porque hay palabras que se repiten en muchos estilos o porque no tenemos suficientes datos para entrenarlo correctamente además de todas las modificaciones necesarias al dataset o los problemas que hemos tenido con Google Collab y el tiempo necesario de ejecución. Aunque la precisión no sea muy alta (23.8%), el proyecto nos sirve para explorar cómo las letras están relacionadas con los géneros musicales.

Resultados de LSTM con GloVe:

```
Test loss: 1.3865471158708846;
Test Accuracy: 23.861607142857142
```

3.5 Aclaraciones

A lo largo de este trabajo nos hemos encontrado con varios problemas que no nos han dejado hacer todas las pruebas que queríamos. Sobre todo con el entorno de Google colab tiene limitaciones y una vez se nos acababa la GPU teníamos que esperar hasta el día siguiente para seguir ejecutando el código. De cara a la próxima entrega, nuestro objetivo es optimizar todos los modelos porque sabemos los errores que tienen y como poder mejorarlos.

3.6 Resumen

Para esta primera iteración, lo que hemos hecho ha sido probar con distintos modelos la clasificación de género. Primero, para el enfoque con shallow hemos probado 5 modelos distintos de los cuales Random Forest y Logistic Regression han sido los que mejores resultados hemos obtenido. Además para estos modelos hemos identificado cuales son las palabras más influyentes. En la lista de palabras del Random Forest podemos ver varias palabras malsonantes las cuales suelen estar muy relacionadas con el género del rap y este género es el que más accuracy tiene con un 88% por lo que es lógico. Para la clasificación con CNN también hemos probado dos arquitecturas y a cada una de ellas hemos activado y desactivado los embeddings de Word2Vec. Esperábamos obtener mejores resultados teniéndolos activados sin embargo la mejora es mínima. Además, para la entrega final hemos añadido las métricas que nos faltaron como recall o f1-score para poder compararlo con el modelo de Transformers final. En cuanto a la clasificación con LSTM, se usaron los embeddings de GLOVE los cuales relacionan las palabras. Para este modelo no obtuvimos buenos resultados y no conseguía diferenciar bien entre géneros.

4. Segunda iteración de resolución del problema - E4

4.1 Clasificación con Transformers

Para la clasificación con Transformers hemos usado DistilBERT. Este modelo tiene como objetivo principal reducir el tamaño de BERT para que pudiese ser desplegado en entornos donde se requieran tiempos de respuesta más rápidos. Básicamente ofrece el mismo rendimiento que BERT pero con un 40% menos de memoria y un 60% y es por eso por lo que hemos escogido este modelo.

4.1.1 Dataset

Respecto al dataset hemos tenido que hacer algunos cambios para que sea compatible con los modelos de Transformers de HuggingFace como cambiar el nombre de la variable a labels para que se reconozca. También hacía falta convertir las etiquetas categóricas en valores numéricos y por último lo convertimos al formato de datasets de HuggingFace usando la clase Dataset.

4.1.2 Modelo utilizado

Como mencionamos anteriormente hemos utilizado DistilBERT concretamente “distilbert-base-uncased” como modelo base debido a su equilibrio entre rendimiento y eficiencia computacional.

Epoch	Training Loss	Validation Loss	Accuracy	F1-Score
1	0.825600	0.775606	0.668000	0.666414
2	0.700300	0.769726	0.670667	0.672635
3	0.534600	0.844629	0.679333	0.677918

Estos son los resultados de las 4 métricas que recopilamos tras 3 epochs. Podemos ver que el modelo da unos resultados razonables con una accuracy máxima de 68% en la tercera iteración.

4.1.3 Comparación de resultados

Una vez ya hemos completado estas dos iteraciones tenemos resultados con distintos métodos o modelos. En primer lugar, tendremos en cuenta el shallow machine learning que nos dio buenos resultados. Después de los modelos de CNN cogeremos el de NLP con embeddings ya que fue el que mejores resultados nos dio. Por último, el LSTM lo descartamos ya que los resultados fueron muy pobres.

Modelo	**Género**	**Precisión**	**Recall**	**f1-score**	**Support**	**Accuracy**
NLP con embeddings	country	0.60	0.69	0.64	1129	59%
	pop	0.45	0.40	0.42	1138	
	rap	0.87	0.88	0.88	1158	
	rock	0.53	0.49	0.51	1075	
Logistic Regression con TfidfVectorizer	country	0.65	0.69	0.67	1129	61%
	pop	0.43	0.36	0.39	1138	
	rap	0.85	0.85	0.85	1158	
	rock	0.49	0.55	0.52	1075	
Distilbert	country	0.476	0.4654	0.4706	1085	68%
	pop	0.8964	0.9116	0.9039	1177	
	rap	0.6141	0.5982	0.606	1120	
	rock	0.7029	0.7236	0.713	1118	

Los resultados obtenidos muestran diferencias claras entre los modelos evaluados en términos de precisión, recall, f1-score y accuracy. De manera general, el modelo DistilBERT destaca al alcanzar un accuracy del 68%, siendo el más alto de los tres enfoques comparados. Este modelo muestra un desempeño sobresaliente en géneros como pop, donde logra casi un 90% en precisión y recall, lo que demuestra una gran capacidad para identificar este género. En contraste, Logistic Regression con TfidfVectorizer obtiene un accuracy del 61%, destacando especialmente en géneros como rap y country, pero mostrando limitaciones evidentes en pop y rock. Por otro lado, el enfoque NLP con embeddings es el que

presenta los resultados más bajos, con un accuracy del 59%, debido principalmente a problemas de recall en géneros como pop, afectando su rendimiento global.

Al analizar los géneros de manera individual, se observa que rap es el más sencillo de clasificar, con buenos resultados en los tres modelos. Sin embargo, géneros como rock y pop presentan mayores desafíos. Aquí, DistilBERT demuestra ser el modelo más equilibrado, logrando un f1-score de 0.7130 en rock y destacando significativamente en pop. Los otros modelos, aunque funcionales, tienen problemas de balance en estas categorías. Este análisis evidencia que, aunque enfoques más simples como Logistic Regression o NLP con embeddings pueden ofrecer buenos resultados iniciales, los modelos basados en Transformers, como DistilBERT, son mucho más efectivos para manejar la complejidad y las particularidades de esta tarea. Por tanto, se recomienda priorizar el uso de DistilBERT en futuras iteraciones del proyecto para maximizar el rendimiento en todos los géneros.

4.2 Generación de texto con GPT-2

Para la nueva tarea con nuestro dataset hemos decidido implementar un modelo de generación de texto en concreto GPT-2. Utilizamos este modelo preentrenado para generar letras de canciones utilizando diferentes métodos de decodificación, evaluando su coherencia y creatividad. Este modelo de lenguaje autorregresivo genera texto prediciendo el siguiente token basándose en los anteriores. En nuestro caso hemos realizado tres enfoques distintos: Greedy Search, Beam Search, y Sampling. Para probarlos vamos usar el prompt "Genre: pop Lyrics: A romantic song about love and heartbreak." que incluye el género musical y una breve descripción

1. Greedy Search
   Texto generado:
   ```
   Lyrics
   . . .
   ,
   I'm a little bit of a romantic, I'm not sure why, but I like to think I can make you feel better. I know you're not the only one who's been hurt
   ```

   El texto comienza con una estructura coherente al principio pero se vuelve un poco repetitivo. Esto es lógico ya que el greedy search selecciona el token con mayor probabilidad lo que hace que tienda a generar texto predecible y repetitivo.

2. Beam Search
   Texto generado:
   ```
   It's about a man and a woman who fall in love, but then they break up, and the song ends with the man saying, "I'll see you in the morning."
   ```

En este caso la respuesta describe una narrativa más clara y coherente. Describe una historia de amor, ruptura y despedida. Lo que hace el Beam Search es explorar múltiples rutas simultáneamente y coge la que es más probable globalmente.

3. Sampling
   Texto generado:
   ```
   The video for "I Just Wanna Be Your Girl" was released on
   November 11, 2016. It was directed by Kamiyama Kenji and
   featured a soundtrack composed by Shiro Sagisu. The video has
   received positive feedback from
   ```

   En el caso del sampling es el que más ha desvariado en cuanto a generación de letras pero aun así tiene cierta lógica y es que el modelo ha generado un texto hablando sobre una canción real de ese género y esa temática.

Por último, hemos probado añadiendo partes de canciones de nuestros datasets como prompts. El problema que hemos visto es que los 3 metodos están generando lo mismo y lo que generan es la continuación de la canción real. Esto ocurre porque GPT-2 está entrenado con estas mismas canciones por lo que genera lo mismo con lo que está entrenado sin importar que método de los tres 3 usemos.

5. BIBLIOGRAFÍA:

- *Tanto las transparencias de ALUD como sus Collabs*
- *ChatGPT*
- *GPT-2*
- *Transformers*
- *Clasificación LSTM*
- *Word2Vec*
- *ELMo*