



Trabajo: **Detección de género musical**

CE Inteligencia Artificial y Big Data
Modelos de Inteligencia Artificial
2024/2025

Daniel Marín López
Guadalupe Luna Velázquez
Marta López Urbano
Víctor Páez Anguita

Índice

1. Introducción	3
2. ¿Qué se pide?	3
3. Cómo abordar el problema.....	3
3.1. Recolección de datos.....	4
3.2. Preprocesamiento de texto.....	6
3.2.1 Limpieza de datos.....	6
3.2.2 Tokenización, Lematización y Stop-Words	9
3.3. Representación del texto	10
3.4. Selección del modelo y entrenamiento	13
3.5. Pruebas	18
3.6. Creación del servidor web.....	20
4. Conclusiones	20

1. Introducción

En esta práctica nos centraremos en el funcionamiento de la minería de texto. Esta disciplina, que se encuentra en la intersección de la lingüística computacional y la inteligencia artificial, busca desentrañar los secretos ocultos en grandes volúmenes de texto no estructurado. A través de técnicas avanzadas de procesamiento del lenguaje natural (NLP), transformaremos este mar de palabras en información valiosa, identificando patrones, tendencias y conocimientos que de otra manera serían invisibles. Desde la detección de spam en correos electrónicos hasta el análisis de opiniones en redes sociales, las aplicaciones de la minería de texto son infinitas y cada vez más relevantes en nuestro mundo digital. En esta práctica, exploraremos los fundamentos teóricos y prácticos de esta disciplina, poniendo en práctica nuestros conocimientos a través de ejercicios y proyectos reales.

2. ¿Qué se pide?

Realizar un sistema que al pasarle una letra de una canción detecte si pertenece a un género o no. En nuestro caso intentaremos que detecte el género pop.

3. Cómo abordar el problema

Para poder abordar el problema se puede hacer de la siguiente forma:

1. **Recolección de datos:** Buscaremos un dataset que podamos usar para nuestro modelo.
2. **Preprocesamiento del texto:** Limpiaremos y normalizaremos las letras, eliminando signos de puntuación, convirtiendo todo a minúsculas y aplicando técnicas de stemming o lematización.
 - Limpieza de datos
 - Tokenización
 - Lematización
 - Stop-Words
3. **Representación del texto:** Convertiremos las letras en representaciones numéricas que puedan ser procesadas por algoritmos de machine learning. Utilizaremos técnicas como Bag-of-Words, TF-IDF o modelos de lenguaje preentrenados (e.g., Word2Vec, BERT).
 - Bag of Words
 - TF-IDF
 - Embeddings
4. **Selección del modelo y su entrenamiento:** Entrenaremos un modelo de clasificación (e.g., Naive Bayes, Support Vector Machine, Redes Neuronales) utilizando un conjunto de datos etiquetado con letras de canciones claramente clasificadas como pop o no pop.
5. **Pruebas:** Evaluaremos el desempeño del modelo utilizando métricas como precisión, recall y F1-score.

3.1. Recolección de datos

El primer paso consiste en recolectar un conjunto de datos que contenga una amplia variedad de letras de canciones, categorizadas por género musical. Este dataset será esencial para entrenar y evaluar nuestro modelo de clasificación, ya que proporcionará ejemplos representativos de canciones pop y de otros géneros.

En este caso, utilizaremos el dataset "Genius Song Lyrics", disponible en [Kaggle](#), que incluye información sobre canciones lanzadas hasta 2022 de la página [Genius](#).

1. Configuración del entorno para descargar los datos:

Para comenzar, necesitamos descargar el dataset desde Kaggle. Es importante configurar correctamente la autenticación para la API de Kaggle, asegurándonos de que el archivo "kaggle.json" esté presente en el directorio adecuado. Este archivo contiene las credenciales necesarias para acceder a los datasets de la plataforma.

```
os.makedirs("/content/data/.kaggle", exist_ok=True)
!mv /content/kaggle.json /content/.kaggle/
!chmod 600 /content/data/.kaggle/kaggle.json
```

2. Descargar el dataset, descomprimir y cargar los datos:

Una vez configurada la autenticación, ejecutamos el comando para descargar el dataset, lo descomprimos y lo cargamos en un DataFrame de Pandas para su manipulación y análisis.

```
# Descarga el dataset
!kaggle datasets download -d carlosgcdj/genius-song-lyrics-with-language-information
with zipfile.ZipFile("genius-song-lyrics-with-language-information.zip", 'r') as zip_ref:
    zip_ref.extractall("/content/data")
```

3. Borrar el archivo zip:

Ya que tenemos descomprimos y cargados los datos, podemos borrar el archivo zip que nos ocupa espacio.

```
# Borramos el archivo zip
os.remove("/content/genius-song-lyrics-with-language-information.zip")
```

Una vez tengamos esto hecho podremos trabajar con nuestro dataframe de información.

4. Extraer las canciones en inglés:

Una vez se han descargado los datos, los guardamos en un DataFrame de Pandas. Debido a la magnitud inmensa de los datos, optamos por filtrar por canciones cuyo idioma sea inglés para que nuestro dataset sea mínimamente procesado.

```
chunk_size = 100000 # Define el tamaño del chunk
chunks = []
for chunk in pd.read_csv('/content/data/song_lyrics.csv', chunksize=chunk_size):
    # Procesar cada chunk (por ejemplo, filtrar por idioma o género)
    chunk = chunk[(chunk['language'] == 'en')]
    chunks.append(chunk)
# Combina los chunks si es necesario
df = pd.concat(chunks)
df
```

Con este código recorreremos el CSV en bloques de 10.000 registros, lo que ayuda a facilitar el procesamiento del mismo y no se resiente demasiado la memoria disponible en el Colab. Luego se seleccionan aquellos registros que tengan el idioma inglés ya que será mucho mejor enfocarnos en un idioma y si es el pop mejor ya que la mayoría de las canciones en este género son en inglés (con más de 1 millón de canciones registradas).

A partir de ahora iremos guardando nuestro progreso en Drive para tenerlos como copia de seguridad y para no tener que gastar almacenamiento en el cuaderno.

```
drive.mount('/content/drive')
ruta = '/content/drive/MyDrive/Google Colab/PIA/data/'
os.makedirs(ruta, exist_ok=True)
ruta_archivo = os.path.join(ruta, 'en_lyrics.csv')
df.to_csv(ruta_archivo, index=False)
```

El código muestra cómo se monta la carpeta de Drive y en la variable 'ruta' indicamos la ruta de la carpeta donde se guardará el archivo, si no existe la librería 'os' la crearía. Por último, se exporta el CSV añadiendo a la ruta anterior el nombre del archivo.

Además para cargar estos archivos lo haremos con el siguiente código, ya que será en más de una ocasión como si fueran checkpoints para guardar el progreso.

```
# Load the preprocessed data
drive.mount('/content/drive')
ruta = '/content/drive/MyDrive/Google Colab/PIA/data/'
ruta_archivo = os.path.join(ruta, 'pop_comb.csv')
df = pd.read_csv(ruta_archivo)
df
```

3.2. Preprocesamiento de texto

El preprocesamiento del dataset es un paso fundamental para garantizar que los datos estén en las mejores condiciones para el análisis y modelado.

3.2.1 Limpieza de datos

Primero para empezar a limpiar nuestro dataset, listamos las columnas que tiene el dataframe.

```
df = df
print(df.info())
```

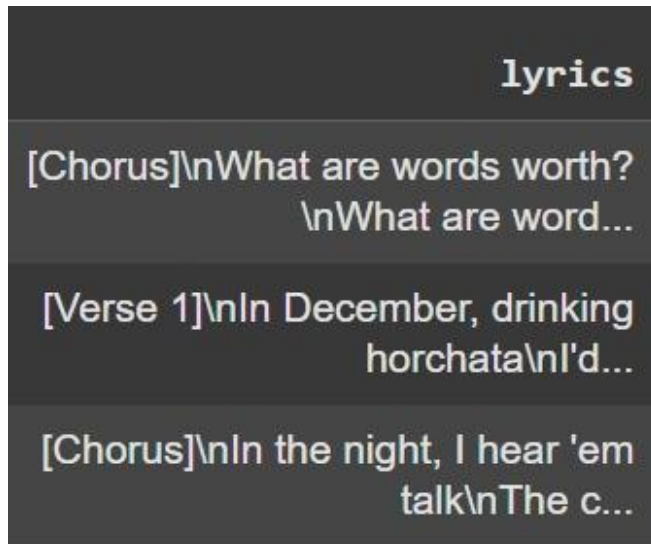
Y se nos muestra lo siguiente.

```
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   title                 59998 non-null  object
1   tag                   60000 non-null  object
2   artist                60000 non-null  object
3   year                  60000 non-null  int64
4   views                 60000 non-null  int64
5   features              60000 non-null  object
6   lyrics                60000 non-null  object
7   id                    60000 non-null  int64
8   language_cld3         60000 non-null  object
9   language_ft           60000 non-null  object
10  language               60000 non-null  object
dtypes: int64(3), object(8)
```

Después de ver sus columnas, seleccionaremos las columnas relevantes, por lo que nos quedamos con las columnas género y la letra, borrando así las demás.

```
del df['title']
del df['features']
del df['views']
del df['year']
del df['artist']
del df['language_cld3']
del df['language']
del df['language_ft']
del df['id']
```

Lo siguiente será limpiar el formato de la letra de las canciones, ya que como vemos tiene símbolos o etiquetas para señalar los saltos de línea y las estrofas.



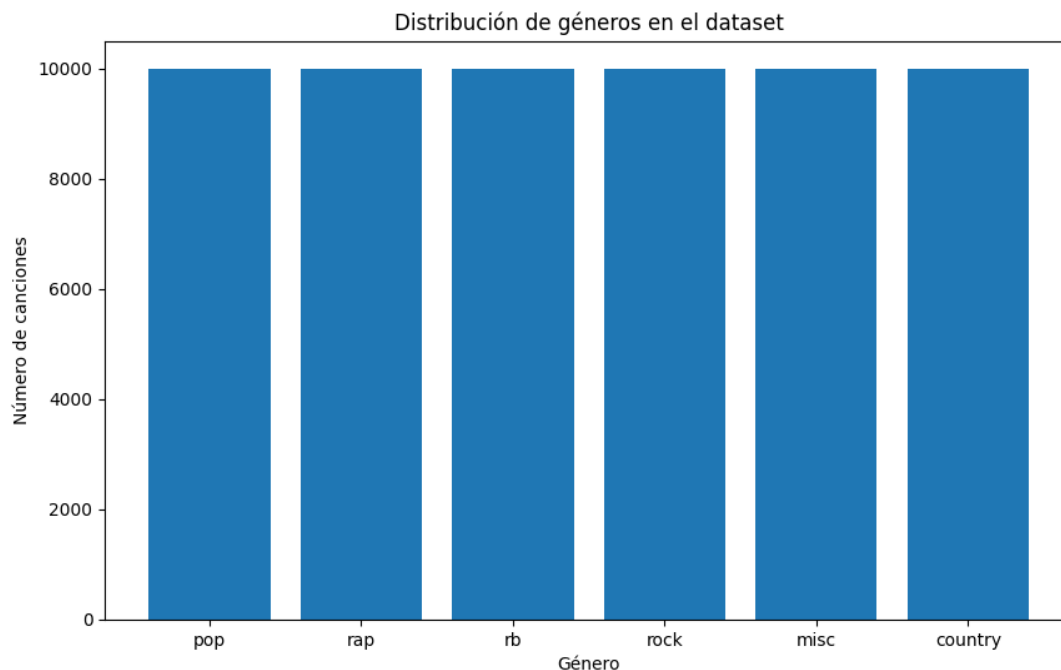
Por lo que usaremos este código que quita las etiquetas, las mayúsculas, los saltos de línea, etc.

```
import re

def preprocess_text(text):
    text = re.sub(r"\.[*?]", "", text)
    text = re.sub(r"\n\s*\n", "\n", text)
    text = text.lower()
    text = re.sub(r"^[a-zA-Z0-9\s]", "", text)
    text = re.sub(r'\s+', ' ', text)
    return text.strip()

df['lyrics'] = df['lyrics'].apply(preprocess_text)
```

Ahora podremos analizar la distribución de géneros musicales en el dataset. En un principio nos habíamos quedado con 10000 canciones de cada género musical.



Sin embargo, nosotros trabajaremos con el género pop que es el que queremos detectar y el resto los cortaremos a 2000 registros cada uno y los reasignaremos a la categoría "non-pop".

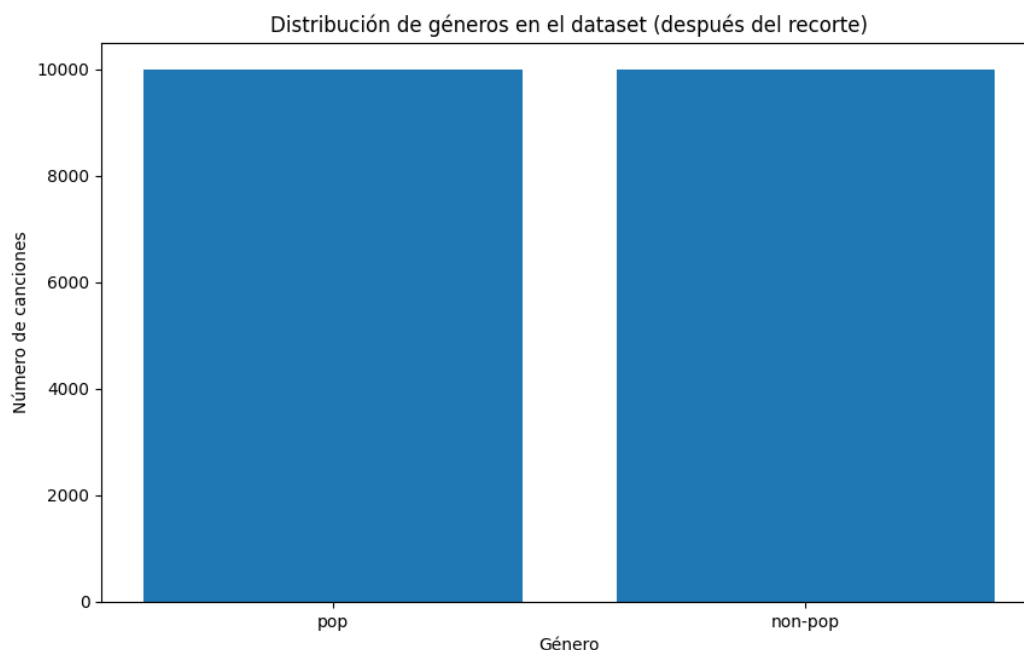
```
# Assuming 'df' is your DataFrame and it has a column named 'tag'
non_pop_genres = df[df['tag'] != 'pop']
max_non_pop = 2000

# Group by genre and sample up to max_non_pop
sampled_non_pop = non_pop_genres.groupby('tag').apply(lambda x: x.sample(min(len(x),
max_non_pop))).reset_index(drop=True)

# Change the tag to 'non-pop' for the sampled data
sampled_non_pop['tag'] = 'non-pop'

# Concatenate with the original pop data
pop_data = df[df['tag'] == 'pop']
new_df = pd.concat([pop_data, sampled_non_pop])
```


Por lo que el dataset nos quedaría así, quedando de una manera balanceada entre la categoría pop y non-pop.



3.2.2 Tokenización, Lematización y Stop-Words

Lo siguiente será procesar el texto resultante de cada letra y extraer las palabras claves que identifican las letras de estilo pop. Podemos usar para realizar esta labor las librerías NLTK y spaCy, nosotros usaremos esta última ya que en caso de incluir canciones de otros idiomas es perfecto ya que tiene distintos paquetes dependiendo del idioma.

```
nlp = spacy.load("en_core_web_sm")

def preprocess_text(text):
    # Tokenización, lematización y eliminación de stop words
    doc = nlp(text)
    tokens = [token.lemma_ for token in doc if not token.is_stop and token.is_alpha]
    return tokens

tokens = df['lyrics'].apply(preprocess_text)
```

Terminado el proceso, se generan en una columna nueva los tokens de las palabras por cada canción. Al principio lo usábamos pero nos dimos cuenta que TF-IDF daba mejores resultados que este.

Sin embargo, hay que tener en cuenta si el modelo estuviera desbalanceado, en nuestro caso ese problema lo resolvimos al crear simplemente 2 clases pop y non-pop y tener el mismo número de registros en ambas.

Se puede hacer 2 cosas para tratar el desbalance:

- Sobremuestreo: Generar datos sintéticos en la clase minoritaria.
- Submuestreo: Borrar datos de la clase mayoritaria, conlleva una pérdida de información.

Este sería el ejemplo anterior de hacer submuestreo.

```
# Get the number of "pop" songs
pop_count = df[df['genre'] == 'pop'].shape[0]

# Randomly sample "non-pop" songs to match the number of "pop" songs
non_pop_df = df[df['genre'] == 'non-pop'].sample(n=pop_count, random_state=42) # Set random_state
for reproducibility

# Concatenate the "pop" songs with the sampled "non-pop" songs
balanced_df = pd.concat([df[df['genre'] == 'pop'], non_pop_df])

# Shuffle the balanced dataset
balanced_df = balanced_df.sample(frac=1, random_state=42).reset_index(drop=True)

# Now 'balanced_df' contains an equal number of 'pop' and 'non-pop' songs
genres = balanced_df["genre"].value_counts()
```

Tras tener en cuenta estos puntos, seguiremos con los siguientes pasos.

3.3. Representación del texto

Una vez que hemos procesado y limpiado nuestro conjunto de datos de letras de canciones, podemos comenzar a construir representaciones vectoriales que permitan a nuestros modelos comprender y analizar el texto. Dos técnicas comunes para lograr esto son el modelo de **Bolsa de Palabras (Bag of Words)** y los **Embeddings**.

El modelo de Bolsa de Palabras es una representación simple pero efectiva. Cada documento (en este caso, cada letra de canción) se representa como un vector donde cada dimensión corresponde a una palabra del vocabulario. El valor de cada dimensión indica la frecuencia con la que esa palabra aparece en el documento. Sin embargo, este modelo no captura el orden de las palabras ni las relaciones semánticas entre ellas.

Para abordar estas limitaciones, podemos utilizar **Embeddings**. Los Embeddings asignan a cada palabra un vector denso de números reales en un espacio vectorial de alta dimensión. Palabras con significados similares tienden a estar cercanas en este espacio. Técnicas como Word2Vec y GloVe son ampliamente utilizadas para aprender estos embeddings.

Además de utilizar embeddings, podemos emplear **TF-IDF** (Term Frequency-Inverse Document Frequency) para ponderar la importancia de las palabras en cada documento. TF-IDF asigna un peso mayor a las palabras que son frecuentes en un documento pero poco frecuentes en el corpus completo. Esto nos permite identificar las palabras clave y distintivas de cada canción o género musical.

Al aplicar TF-IDF a nuestro conjunto de datos de canciones pop, podemos descubrir qué palabras son más características de este estilo musical. Por ejemplo, podríamos encontrar que términos como "amor", "corazón", "bailar" y "fiesta" aparecen con mayor frecuencia en las letras de canciones pop en comparación con otros géneros. Esta información puede ser útil para tareas como clasificación de géneros, recomendación de canciones o análisis de temas."

Podemos buscar los 10 términos más utilizados en las canciones de género pop. Esto nos puede dar una idea de qué palabras son comunes en este género que se pueden luego usar para que nuestro modelo preste más atención a estas palabras a la hora de clasificar. Usaremos la librería `TfidfVectorizer` para realizar este trabajo:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Vectorizador TF-IDF
vectorizer = TfidfVectorizer(max_features=1000, stop_words='english', ngram_range=(1, 2))
X = vectorizer.fit_transform(df['lyrics'])
feature_names = vectorizer.get_feature_names_out()

# Promedio de TF-IDF por clase
pop_tfidf = X[df['genre'] == 'pop'].mean(axis=0).A1
other_tfidf = X[df['genre'] != 'pop'].mean(axis=0).A1

# Diferencia de importancia
tfidf_diff = pop_tfidf - other_tfidf
relevant_features = sorted(zip(tfidf_diff, feature_names), reverse=True)

print("Palabras o n-gramas más relevantes para 'pop':")
for diff, feature in relevant_features[:10]:
    print(f"{feature}: {diff}")
```

El resultado es el siguiente:

```
Palabras o n-gramas más relevantes para 'pop':
youre: 0.006729286882871496
love: 0.005139785544598843
oh: 0.004785013607103348
away: 0.004380713439729816
chorus: 0.0035032682182011493
theres: 0.00333731223611387
ill: 0.003239090704028537
feel: 0.0030233785330298843
heart: 0.0028414819764453897
oh oh: 0.002793368545968348
```

El análisis de frecuencia de palabras en las letras de canciones pop revela patrones interesantes. Términos como 'love' y 'away' ocupan posiciones destacadas, sugiriendo una fuerte carga emocional y una búsqueda de conexión en este género. A diferencia de palabras más genéricas como 'heart' o 'feel', estos términos connotan sentimientos profundos y experiencias personales. Esta información puede ser valiosa para nuestro modelo de IA, no solo para clasificar canciones sino también para identificar subgéneros o incluso predecir tendencias en la composición de letras.

También podemos hacer una nube de palabras que muestre no solo las 10 palabras más frecuentes, sino ver de manera gráfica más palabras que también tienen presencia en las canciones de género pop.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Create a dictionary of words and their importance scores
word_freq = {feature: diff for diff, feature in relevant_features[:50]} # Take top 50 for clarity

# Generate the word cloud
wordcloud = WordCloud(width=800, height=400,
background_color='white').generate_from_frequencies(word_freq)

# Display the word cloud
plt.figure(figsize=(10, 5), facecolor=None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```

Cuando queremos que un ordenador “entienda” el lenguaje natural, necesitamos transformar las palabras en números. Esto es como traducir un idioma a otro, pero en vez de usar palabras, usamos números. Una de las formas más populares de hacer esto es utilizando modelos como Word2Vec.

12

frecuencia, así que Word2Vec asignará a ambas palabras vectores numéricos muy similares. De esta manera, el ordenador puede entender que “perro” y “gato” son conceptos relacionados.

En este caso, hemos utilizado Word2Vec para transformar los tokens que identificamos previamente en vectores numéricos. Estos vectores capturan el significado y contexto de cada token, permitiéndonos realizar análisis más sofisticados.

```
from gensim.models import Word2Vec

def train_word2vec_model(sentences, vector_size=100, window=5, min_count=1, epochs=10):
    model = Word2Vec(sentences=sentences, vector_size=vector_size, window=window,
min_count=min_count, epochs=epochs)
    return model

sentences = df['tokens'].apply(lambda x: x.strip('[]').replace("'", "").split(',')).tolist()
model = train_word2vec_model(sentences)
```

De esta forma, tendríamos nuestro modelo Word2Vec que transformaría de las palabras a un vector numérico. Para ellos hacemos el siguiente código:

```
word_vector = model.wv["love"]
print("Vector para 'love':", word_vector)

-----
Vector para 'love': [ 1.5421209  -1.057031   0.06016188  0.9686033  0.47728133 -2.5916092
 0.6604153  0.38008192 -0.4050451  -1.4620706  0.01230923 -2.5738866
 1.3092611  0.3803202  0.63216865  1.8411928  0.5664323  0.5389144
-1.2584234  0.73957556 -0.17732283  0.7142853  -1.6885821  -0.23074003
-0.53766567  0.9771871  0.67522466  0.3035593  -0.6068045  1.5958049
-1.0368469  -0.6108287  0.15926051 -1.2223814  -1.2256604  2.4666402
-0.9308293  -1.3514848  -1.8934891  -2.5792499  -0.24301313  1.4152671
 0.44767037 -0.20210703  0.4405086  -0.30869687  0.97072333 -1.2860585
 1.0824164  0.8388727  0.6827698  -0.05902986  1.3612216  -0.47763738
 0.86536586  0.8580091  1.4328394  -0.38854218  1.4963969  1.9419416
-0.54207754 -1.8025631  0.52928716 -2.330107  -0.8284404  3.1633542
-1.9180167  2.3457844  0.6423778  -0.4456087  -1.6296196  -0.4057187
-1.7029157  0.02103424 -0.13207938  0.5535433  0.3252702  0.32164028
-0.15184537 -2.8128192  -0.0548684  -0.3636386  0.92326283  2.5116413
 1.5984597  0.9232482  -0.6830471  -0.10881469  1.0059901  0.32241794
 0.97663325 -2.1665213  -0.50878817 -0.25764084  0.1370399  0.3558743
 1.2819608  0.05530247 -0.2275207  -1.2481772 ]
```

Usando este modelo de Word2Vec, podemos convertir las palabras a vectores numéricos que luego nuestro modelo usaría para clasificar las canciones según el género si pertenecen al pop o no.

3.4. Selección del modelo y entrenamiento

La elección del modelo adecuado es crucial en cualquier proyecto de aprendizaje automático. Entre las opciones más populares encontramos la Regresión Logística, ideal para problemas de clasificación binaria. Las Redes Neuronales, como las Secuenciales, ofrecen una mayor flexibilidad y capacidad de aprendizaje, especialmente en tareas que

involucran datos secuenciales. El clasificador SGD (Stochastic Gradient Descent) es una variante de las redes neuronales que utiliza una técnica de optimización eficiente. Cada uno de estos modelos presenta ventajas y desventajas específicas, por lo que la elección dependerá de la naturaleza de los datos, el problema a resolver y los recursos computacionales disponibles.

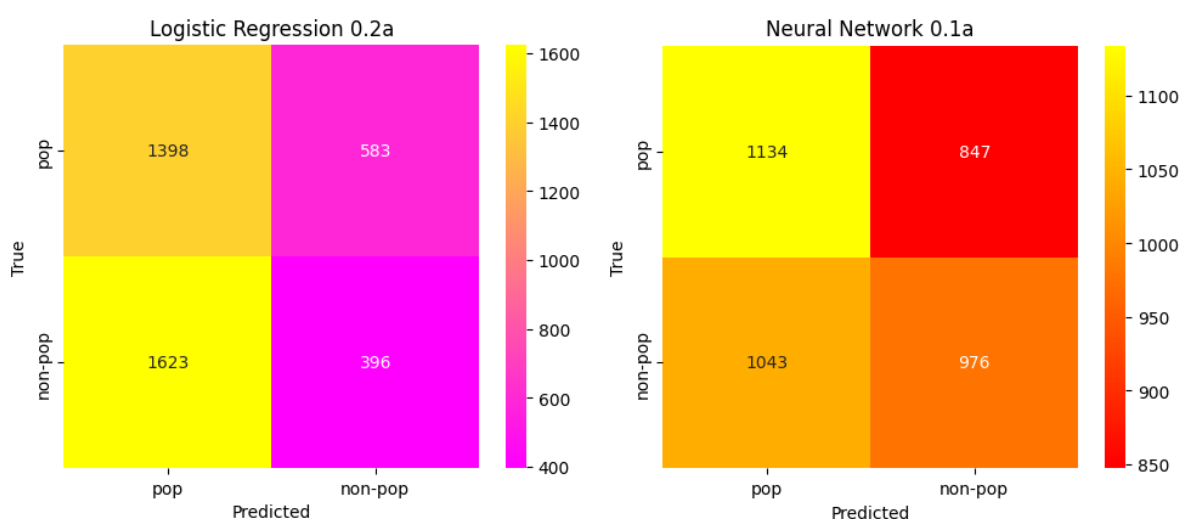
Anteriormente usábamos el siguiente código:

```
# Function to create document vectors
def document_vector(tokens, model, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0
    for word in tokens:
        try:
            vec += model.wv[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary
            continue
    if count != 0:
        vec /= count
    return vec

# Generate document vectors
document_vectors = []
for tokens in df['tokens']:
    document_vectors.append(document_vector(tokens, model, 100))

# Convert document vectors to a NumPy array
document_vectors = np.concatenate(document_vectors)
```

El código genera un vector Numpy que recoge todas las palabras y pasa por el modelo de Word2Vec que creamos anteriormente. Sin embargo los modelos no daban buenos resultados como se puede apreciar en las matrices de confusión:



Como se observa en los resultados, los modelos de clasificación tienden a favorecer significativamente el género pop, lo que sugiere un sesgo en el conjunto de datos o en las características utilizadas para la clasificación. Esta tendencia puede deberse a la mayor representatividad del pop en las bases de datos musicales o a que las características extraídas de las canciones pop son más distintivas y fáciles de identificar. Para abordar este problema y mejorar la precisión de la clasificación en otros géneros musicales, se implementó una nueva estrategia basada en el uso de TF-IDF. Esta técnica

permite ponderar la importancia de las palabras clave en función de su frecuencia en cada género, lo que ayuda a resaltar las características distintivas de cada uno y a reducir el sesgo hacia el género pop. El código puede ser de la siguiente forma:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import classification_report

tfidf_df = pd.DataFrame(X_tf.toarray(), columns=vectorizer.get_feature_names_out())

# Prepare the data for training
X = tfidf_df
y = df['tag']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a logistic regression model
clf = LogisticRegression(max_iter=1000, C=1.0, penalty='l1', solver='liblinear')
clf3 = SGDClassifier(loss='hinge', penalty='l2', alpha=1e-3, random_state=42, max_iter=5, tol=None)

clf.fit(X_train, y_train)
clf3.partial_fit(X_train, y_train, classes=np.unique(y))

# Predict on the test set
y_pred = clf.predict(X_test)
y_pred3 = clf3.predict(X_test)

# Evaluate the model
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("="*35)
print("\nClassification Report 3:\n", classification_report(y_test, y_pred3))
```

Primero hacemos un dataframe sobre el TF-IDF en donde convertimos la matriz densa a un array porque, por eficiencia, está dispersa y luego para las columnas usamos los términos identificados en por el TF-IDF en donde obtenemos que cada fila es una letra y cada palabra una columna. Tras eso, separamos los datos de entrenamiento y prueba y vamos probando diferentes modelos. Aquí hemos probado Regresión Logística y SGD Classifier con distintos hiperparámetros para probar su desempeño.

El código de la red neuronal es el siguiente:

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import LabelEncoder

# Encode labels to numerical values (0 and 1)
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

input_dim = X_train.shape[1]

clf2 = Sequential([
    Dense(128, activation='relu', input_shape=(input_dim,)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1, activation='sigmoid') # Salida binaria (pop o no pop)
])
clf2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Use encoded labels for training
clf2.fit(np.array(X_train), np.array(y_train_encoded), epochs=10, batch_size=32,
        validation_split=0.2)

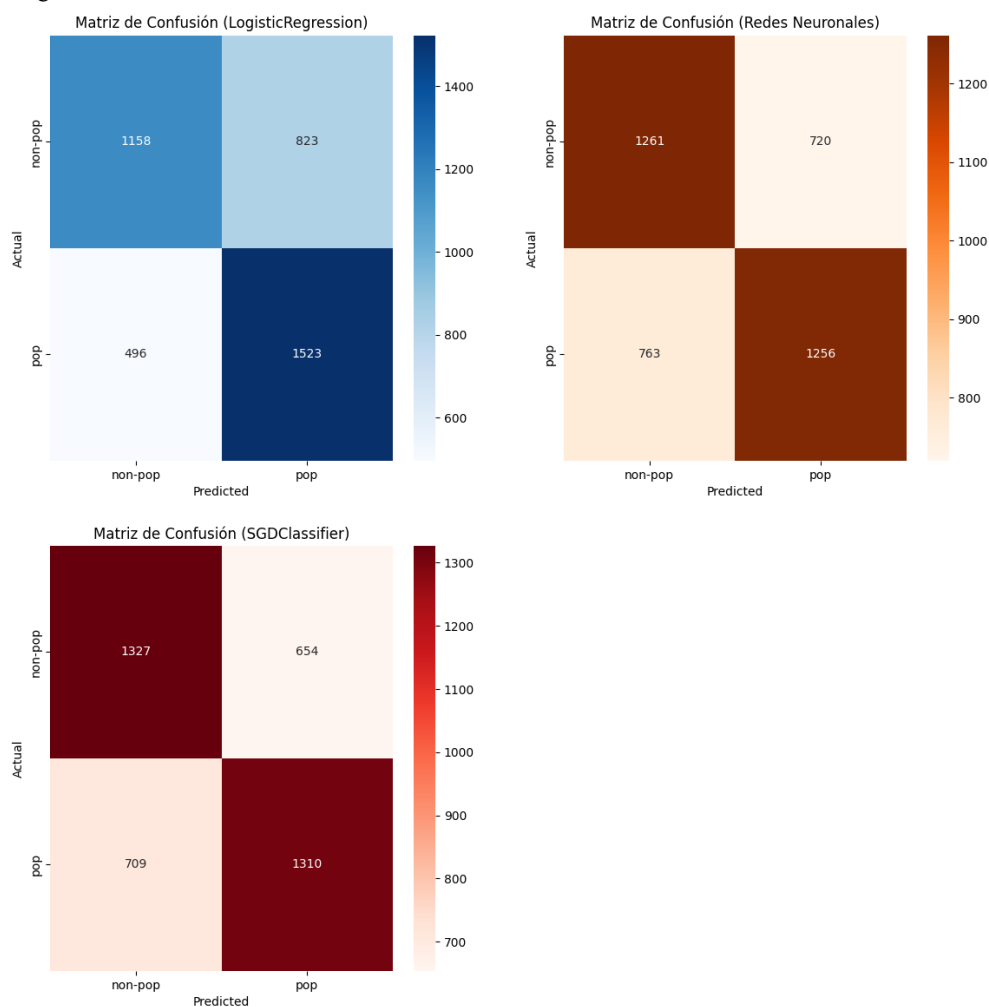
# Predict and evaluate using encoded labels
y_pred2 = (clf2.predict(np.array(X_test)) > 0.5).astype("int32")
y_pred2_labels = label_encoder.inverse_transform(y_pred2.flatten())
print("\nClassification Report:\n", classification_report(y_test, y_pred2_labels))
```

Primero le generamos unos labels binarios para las salidas, luego se define una red neuronal secuencial para una tarea de clasificación binaria. Esta red tiene tres capas densamente conectadas, cada una con un número específico de neuronas. La primera capa recibe la entrada de datos, mientras que las dos siguientes procesan esta información. Las funciones de activación ReLU introducen no linealidad en las capas ocultas, permitiendo a la red aprender patrones complejos. Las capas de dropout ayudan a prevenir el sobreajuste al desactivar aleatoriamente un porcentaje de neuronas durante el entrenamiento. La capa de salida utiliza una función de activación sigmoide para producir una probabilidad entre 0 y 1, indicando la probabilidad de que la entrada pertenezca a una de las dos clases posibles (por ejemplo, "pop" o "no pop"). El tamaño de la entrada (input_dim) y el número de neuronas en cada capa pueden ajustarse según las características del conjunto de datos y el problema específico.

Luego se configura la red neuronal para resolver nuestro problema. Utiliza el optimizador **Adam**, que ajusta los pesos adaptativamente para minimizar el error; la función de pérdida **binary_crossentropy**, ideal para medir discrepancias en problemas con salidas binarias (0 o 1); y la métrica **accuracy** para evaluar el porcentaje de predicciones correctas durante el entrenamiento y la validación.

Por último hacemos el entrenamiento con 20 epochs, es decir, cada valor de entrenamiento pasará 20 veces por la red neuronal.

El resultado es el siguiente:



Al analizar las matrices de confusión, observamos una mejora significativa en el desempeño del modelo tras el cambio de enfoque en el conjunto de datos. Esta evolución positiva se atribuye principalmente al dataset que al parecer ha arrojado mejores resultados que el anterior, además de normalizar y balancear el dataset para ajustarlo a nuestro caso de uso. Estos ajustes han permitido al modelo capturar patrones más sutiles y generalizar mejor a nuevos datos.

3.5. Pruebas

Vamos a realizar las pruebas ya sobre nuestros modelos usando el siguiente código donde nos pedirá que insertemos la letra de la canción.

```
def classify_song():
    lyrics = input("Introduce las lyrics de la canción: ")

    # Vectoriza las lyrics usando el TfidfVectorizer
    lyrics_tfidf = vectorizer.transform([lyrics])

    # Predict using the logistic regression model
    prediction_lr = clf.predict(lyrics_tfidf)[0]

    # Predict using the neural network model
    prediction_nn = (clf2.predict(lyrics_tfidf) > 0.5).astype("int32")[0][0]
    prediction_nn = label_encoder.inverse_transform([prediction_nn])[0]

    # Predict using the SGD Classifier model
    prediction_sgd = clf3.predict(lyrics_tfidf)[0]

    print(f"Predicción con Regresión Logística: {prediction_lr}")
    print(f"Predicción con Red Neuronal: {prediction_nn}")
    print(f"Predicción con SGD Classifier: {prediction_sgd}")

classify_song()
```

Probaremos con diferentes géneros con los que se ha entrenado, como por ejemplo rap, con la canción Hypnotize, en la que le hemos puesto un trozo de la canción y los tres modelos aciertan al decir que no es pop.

```
Introduce las lyrics de la canción: Ha, sicker than your average Poppa twist cabbage off instinct Niggas don't think shit
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names
  warnings.warn(
1/1 ————— 0s 73ms/step
Predicción con Regresión Logística: non-pop
Predicción con Red Neuronal: non-pop
Predicción con SGD Classifier: non-pop
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names
  warnings.warn(
```

Podemos probar con rock, aunque este género suele tener problemas al diferenciarse del pop, usaremos la canción November Rain, en este caso, los modelos consiguen clasificar bien.

```
Introduce las lyrics de la canción: When I look into your eyes I can see a love restrained But darlin' when I hold you Don't you know I f
1/1 ————— 0s 46ms/step
Predicción con Regresión Logística: non-pop
Predicción con Red Neuronal: non-pop
Predicción con SGD Classifier: non-pop
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Logistic
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but SGDClassi
  warnings.warn(
```

Sin embargo ahora vamos a probar con Bohemian Rhapsody, donde se puede ver que acierta la mayoría, pero la red neuronal se equivoca, aunque también hay que tener en cuenta que los resultados pueden cambiar dependiendo de la sesión del Google Colab.

```
Introduce las lyrics de la canción: Mama, just killed a man Put a gun against his head, pulled my trigger, now he's dead Mama, life had just begun But
1/1 ————— 0s 43ms/step
Predicción con Regresión Logística: non-pop
Predicción con Red Neuronal: pop
Predicción con SGD Classifier: non-pop
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRegression was
warnings.warn(
```

Otra prueba será la canción Umbrella de Rihanna, que todos los modelos aciertan.

```
Introduce las lyrics de la canción: No clouds in my stones Let it rain, I hydroplane in the bank Comin' down like Dow Jones when the clouds come.
1/1 ————— 0s 45ms/step
Predicción con Regresión Logística: pop
Predicción con Red Neuronal: pop
Predicción con SGD Classifier: pop
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRegression
warnings.warn(
```

También hemos probado la canción Dark Horse de Katy Perry que al igual que la otra aciertan todos los modelos.

```
Introduce las lyrics de la canción: I knew you were You were gonna come to me And here you are But you better choose carefully 'Cause I, I'm capable of anything
1/1 ————— 0s 45ms/step
Predicción con Regresión Logística: pop
Predicción con Red Neuronal: pop
Predicción con SGD Classifier: pop
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRegression was fitted w
warnings.warn(
```

Ahora vamos a probar una canción llamada All You Ever Do Is Bring Me Down que está clasificada como Country y sin embargo, los tres modelos se equivocan.

```
Introduce las lyrics de la canción: I can't sleep a wink anymore Ever since you first walked out the door Then I just started drinking to forget But I don't think the wors
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature
warnings.warn(
1/1 ————— 0s 75ms/step
Predicción con Regresión Logística: pop
Predicción con Red Neuronal: pop
Predicción con SGD Classifier: pop
```

Seguiremos probando con canciones aleatorias, en este caso es pop, la canción se llama Gravity Hurts y consigue clasificarla como pop.

```
Introduce las lyrics de la canción: Tension is rising Gravity hurts Everything's fallin' apart Choosin' the right side Choosin' our faith
1/1 ————— 0s 47ms/step
Predicción con Regresión Logística: pop
Predicción con Red Neuronal: pop
Predicción con SGD Classifier: pop
```

Para tener más variedad, probaremos otra de rap, llamada Jimmy Cooks, que aciertan todos los modelos.

```
Introduce las lyrics de la canción: Life is only thing we need They need me to go, but I don't wanna leave Rest in peace to Lil Keed Fuck
1/1 ————— 0s 43ms/step
Predicción con Regresión Logística: non-pop
Predicción con Red Neuronal: non-pop
Predicción con SGD Classifier: non-pop
```

Otra de country que hemos probado es Black like me, esta canción a diferencia de la otra la country es clasificada correctamente por todos los modelos.

```

Introduce las lyrics de la canción: Little kid in a small town I did my best just to fit in Broke my heart on the playground, mm When they
1/1 0s 42ms/step
Predicción con Regresión Logística: non-pop
Predicción con Red Neuronal: non-pop
Predicción con SGD Classifier: non-pop

```

Por último probamos otra de country llamada Whiskey Lullaby también clasificada bien por los tres modelos.

```

Introduce las lyrics de la canción: She put him out Like the burning end of a midnight cigarette She broke his heart He spent his whole li
1/1 0s 46ms/step
Predicción con Regresión Logística: non-pop
Predicción con Red Neuronal: non-pop
Predicción con SGD Classifier: non-pop

```

3.6. Creación del servidor web

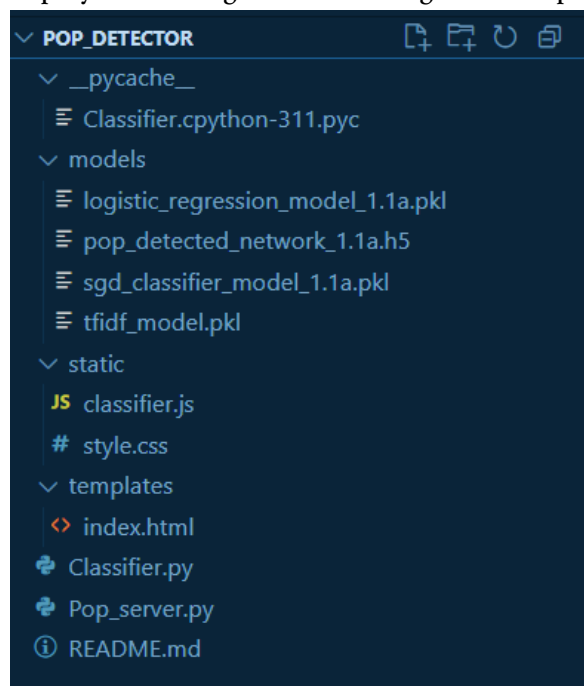
Descripción General

Hemos creado una aplicación web diseñada para clasificar si una canción pertenece al género "pop" basándose en su letra. La aplicación utiliza modelos de aprendizaje automático preentrenados para realizar esta clasificación. Los usuarios pueden ingresar la letra de una canción y obtener predicciones de tres modelos diferentes: Regresión Logística, SGD Classifier y una Red Neuronal.

La aplicación está construida con Flask en el backend y utiliza HTML, CSS y JavaScript para la interfaz de usuario. Los modelos de aprendizaje automático se entrenan previamente y se guardan en archivos para su uso en tiempo real.

Estructura del Proyecto

El proyecto está organizado en las siguientes carpetas y archivos:



models/: Contiene los modelos preentrenados y el vectorizador TF-IDF.

static/: Incluye los archivos CSS y JavaScript para la interfaz de usuario.

templates/: Contiene la plantilla HTML para la página principal.

Classifier.py: Lógica de clasificación.

Pop_server.py: Servidor Flask que maneja las solicitudes y respuestas.

README.md: Documentación del proyecto.

Funcionalidades Principales

Clasificación de Letras de Canciones:

Los usuarios pueden ingresar la letra de una canción en un formulario.

La aplicación utiliza tres modelos para predecir si la canción es "pop" o "non-pop".

Los resultados se muestran en la interfaz con colores que indican la clasificación.

Interfaz de Usuario:

Un formulario para ingresar la letra de la canción.

Botones para clasificar y borrar la información.

Sección para mostrar los resultados de los tres modelos.

Validación de Entrada:

Si el usuario intenta clasificar sin ingresar una letra, se muestra un mensaje de alerta.

Borrado de Información:

El usuario puede borrar la letra ingresada y los resultados mostrados para comenzar de nuevo.

Ahora trataremos sobre algunos aspectos más importantes de nuestro código

Clase Classifier.py

En esta parte cargamos los modelos preentrenados y el vectorizador necesarios para clasificar las letras de las canciones

```
# Cargamos los modelos
with open('models/tfidf_model.pkl', 'rb') as f:
    model_tfidf = pickle.load(f)

with open('models/logistic_regression_model_1.1a.pkl', 'rb') as f:
    modelo_logistico = pickle.load(f)

with open('models/sgd_classifier_model_1.1a.pkl', 'rb') as f:
    modelo_sgd = pickle.load(f)

modelo_nn = keras.models.load_model('models/pop_detected_network_1.1a.h5')
```

Función classify_song()

La función `classify_song` es el núcleo de la aplicación Pop Detector. Su objetivo es clasificar una canción como "pop" o "non-pop" basándose en su letra. Utiliza tres modelos de aprendizaje automático preentrenados para realizar esta tarea.

Proceso

Vectorización:

La letra de la canción se convierte en un formato numérico utilizando un vectorizador TF-IDF. Este paso es esencial porque los modelos de aprendizaje automático no pueden procesar texto directamente.

Clasificación:

La letra vectorizada se pasa a través de tres modelos: Regresión Logística, SGDClassifier, Red Neuronal.

Cada modelo realiza una predicción independiente sobre si la canción es "pop" o "non-pop".

Formateo de Resultados:

Los resultados de los modelos se convierten en un formato legible (por ejemplo, "Pop" o "Non-pop").

Los resultados se devuelven en un diccionario que contiene las predicciones de cada modelo.

```
Tabnine | Edit | Test | Explain | Document
def classify_song(lyrics: str) -> dict:
    # Vectorizar la letra de la canción usando el TfidfVectorizer
    lyrics_vectorized = model_tfidf.transform([lyrics])

    # Clasificación con cada modelo
    resultado_sgd = modelo_sgd.predict(lyrics_vectorized)[0]
    resultado_nn = modelo_nn.predict(lyrics_vectorized.toarray())[0][0] # Convertir a float
    resultado_logistico = modelo_logistico.predict(lyrics_vectorized.toarray())[0]

    # Convertir los resultados a tipos nativos de Python
    resultado_nn = "Pop" if resultado_nn >= 0.5 else "Non-pop" # Convertir a string
    resultado_sgd = str(resultado_sgd) # Convertir a string
    resultado_logistico = str(resultado_logistico) # Convertir a string

    return {
        'SGDClassifier': resultado_sgd,
        'Red Neuronal': resultado_nn,
        'Regresión Logística': resultado_logistico
    }

app = Flask(__name__)
```

Clase Pop_server.py

Función `index()`

Esta función se encarga de mostrar la página principal de la aplicación cuando un usuario visita la ruta raíz (/).

Renderiza la plantilla HTML (`index.html`) que contiene el formulario para ingresar la letra de la canción.

```

from flask import Flask, render_template, request, jsonify
from Classifier import classify_song
import os

app = Flask(__name__)

home = "index.html"

# Ruta principal para mostrar el formulario
@app.route('/')
def index():
    return render_template(home)

```

Función classify()

Procesa la letra enviada por el usuario y la clasifica utilizando los modelos preentrenados.

Devuelve los resultados en formato JSON para que el frontend los muestre al usuario.

Incluye validaciones y manejo de errores para garantizar un funcionamiento robusto.

```

@app.route('/classify', methods=['POST'])
def classify():
    try:
        if request.method == 'POST':
            song_lyrics = request.form.get('lyrics') # Usar get para evitar KeyError
            if not song_lyrics:
                return jsonify({"error": "No lyrics provided"}), 400 # Error si no se envían letras

            # Llamar al modelo de clasificación para obtener resultados de todos los modelos
            resultados = classify_song(song_lyrics)

            return jsonify(resultados)
    except Exception as e:
        # Capturar excepciones y devolver un mensaje de error con detalles
        return jsonify({"error": str(e)}), 500

if __name__ == '__main__':
    app.run(debug=True)

```

Interfaz

En esta imagen podemos visualizar como se ve nuestro servidor web.

Pop Detector

Ingresa la letra de la canción aquí...

Clasificar Borrar

Desarrolladores: Daniel Marín López | Guadalupe Luna Velázquez | Marta López Urbano | Víctor Páez Anguita

Para probar su funcionamiento introducimos la letra de una canción y este lo clasifica basándose en su letra. Como podemos ver se muestra los resultados de tres modelos diferentes y permite al usuario ver cómo cada uno interpreta la letra.

Pop Detector

Mmm, mmm, ah-mmm
Mmm, mmm, mmm

When did it end? All the enjoyment
I'm sad again, don't tell my boyfriend
It's not what he's made for
What was I made for?

'Cause I, 'cause I
I don't know how to feel
But I wanna try
I don't know how to feel
But someday I might
Someday I might

Think I forgot how to be happy
Something I'm not, but something I can
be
Something I wait for
Something I'm made for
Something I'm made for

Clasificar Borrar

Regresión Logística: pop

SGD Classifier: non-pop

Red Neuronal: non-pop

Desarrolladores: Daniel Marín López | Guadalupe Luna Velázquez | Marta López Urbano | Víctor Páez Anguita

Transferencia de audio a texto

Como añadido hemos usado la librería `speech_recognition` para que un usuario pueda subir directamente un archivo de audio y que lo transcriba a texto para poder clasificarlo.

```
Pop_server.py  AudioRecognizerTS.py X
AudioRecognizerTS.py > ...
Victor Paez Anguita, hace 10 minutos | 1 author (Victor Paez Anguita)
1 import os      Victor Paez Anguita, hace 10 minutos • audio reconizer
2 import speech_recognition as sr
3 from flask import Flask, render_template, request, jsonify
4
5
6 app = Flask(__name__)
7
8 app.config['UPLOAD_FOLDER'] = 'uploads' # Carpeta donde se guardarán los audios
9
10 # Asegurarse de que la carpeta de uploads existe
11 if not os.path.exists(app.config['UPLOAD_FOLDER']):
12     os.makedirs(app.config['UPLOAD_FOLDER'])
13
14 def transcribe_audio(audio_path):
15     if not os.path.exists(audio_path):
16         return "El archivo de audio no existe. Por favor, verifica la ruta y vuelve a intentarlo."
17
18     recognizer = sr.Recognizer()
19     try:
20         with sr.AudioFile(audio_path) as source:
21             audio_data = recognizer.record(source)
22             try:
23                 # Cambiar el código de idioma a uno válido
24                 text = recognizer.recognize_google(audio_data, language="en-US")
25                 print("Texto transcrito:", text) # Depuración
26                 return text
27             except sr.UnknownValueError:
28                 return "No se pudo reconocer el audio. Por favor, intenta con un archivo más claro."
29             except sr.RequestError as e:
30                 return f"Error con el servicio de reconocimiento de voz: {e}"
31     except (FileNotFoundError, ValueError) as e:
32         return f"Error al procesar el archivo de audio: {e}"
33
```

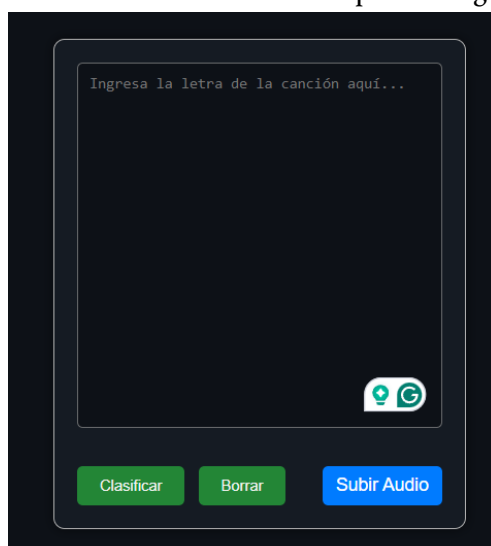
Como podemos observar en el archivo `py` la función `transcribe_audio` hace uso de la clase `Recognizer()` que se encargará de recoger toda la letra detectada en el audio para pasarla a una variable y de este modo pasarla al formulario para transcribir la letra.

```
Pop_server.py > ...
18 def classify():
21     song_lyrics = request.form.get('lyrics') # Usar get para evitar KeyError
22     if not song_lyrics:
23         return jsonify({"error": "No lyrics provided"}), 400 # Error si no se envían letras
24
25     # Llamar al modelo de clasificación para obtener resultados de todos los modelos
26     resultados = classify_song(song_lyrics)
27
28     return jsonify(resultados)
29 except Exception as e:
30     # Capturar excepciones y devolver un mensaje de error con detalles
31     return jsonify({"error": str(e)}), 500
32
33 @app.route('/upload', methods=['POST'])
34 def upload_audio():
35     if 'audio' not in request.files:
36         return jsonify({'error': 'No se ha subido ningún archivo.'})
37
38     file = request.files['audio']
39     if file.filename == '':
40         return jsonify({'error': 'Nombre de archivo vacío.'})
41
42     filename = secure_filename(file.filename)
43     file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
44     print(f"Guardando archivo en: {file_path}") # Depuración
45     file.save(file_path)
46
47     # Transcribir el audio a texto
48     lyrics = transcribe_audio(file_path)
49
50     # Verificar si la transcripción devolvió un error
51     if isinstance(lyrics, str) and lyrics.startswith("Error"):
52         return jsonify({'error': lyrics}), 400 # Devolver el error como respuesta
53
54     # Clasificar la letra transcrita
55     resultados = classify_song(lyrics)
56
57     return jsonify({'letra': lyrics, 'resultados': resultados})
58
59 if __name__ == '__main__':
60     app.run(debug=True)
```

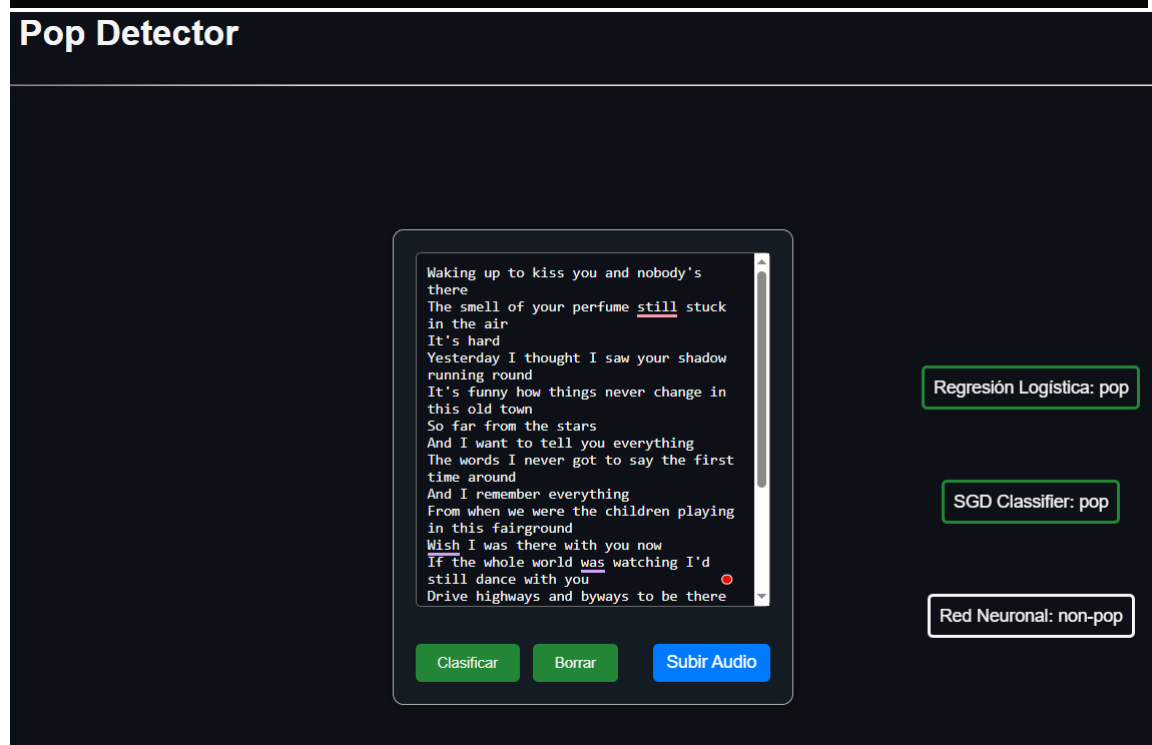
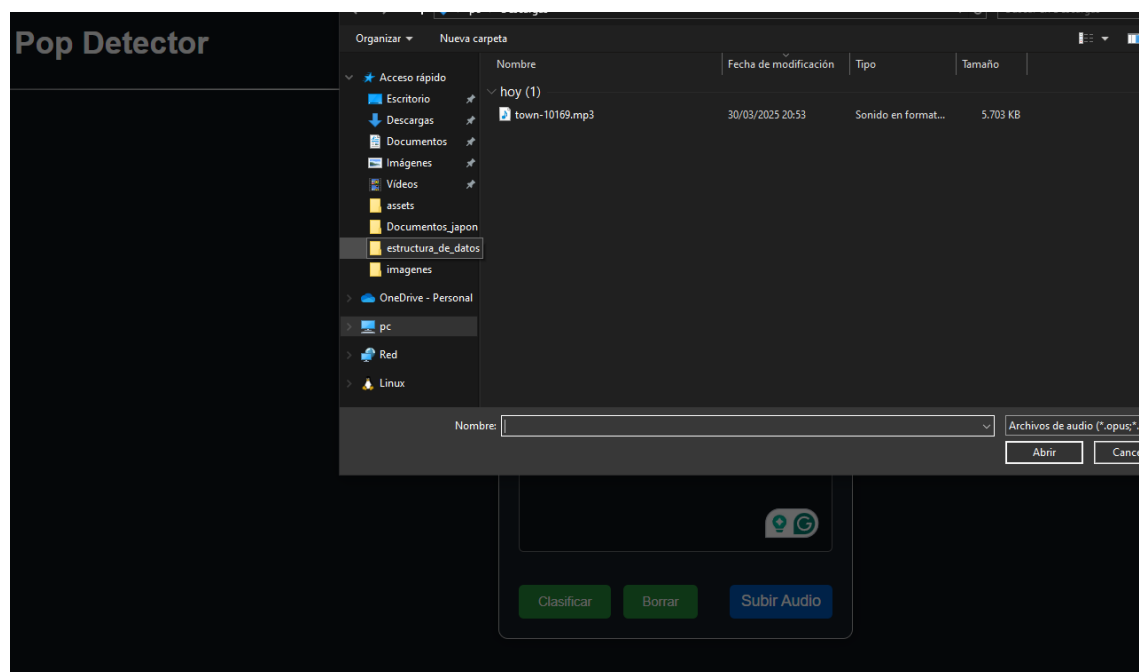
En Pop_Server se ha añadido una nueva ruta para recoger ese audio.

```
83  async function uploadAudio() {
84      const audioFile = document.getElementById('audioFile').files[0];
85
86      const formData = new FormData(document.querySelector('form'));
87      formData.append('audio', audioFile);
88
89      try {
90          // Realiza la solicitud POST a /upload con el archivo de audio
91          const response = await fetch('/upload', {
92              method: 'POST',
93              body: formData
94          });
95
96          // Verifica si la respuesta es correcta antes de intentar parsear el JSON
97          if (!response.ok) {
98              const errorText = await response.text(); // Intenta obtener el texto del error
99              throw new Error(`Error en la solicitud: ${errorText}`);
100          }
101
102          const result = await response.json();
103
104          // Inserta la letra transcrita en el campo de texto del formulario
105          const lyricsTextarea = document.querySelector('textarea[name="lyrics"]');
106          if (lyricsTextarea.value.trim() !== "") {
107              lyricsTextarea.value += "\n\n" + result.letra;
108          } else {
109              lyricsTextarea.value = result.letra;
110          }
111          alert("La transcripción del audio se ha completado y se ha insertado en el formulario.");
112      } catch (error) {
113          // Manejo de errores
114          console.error('Error:', error.message);
115          alert('Hubo un error al procesar el archivo de audio: ' + error.message);
116      }
117  }
```

Y en nuestro archivo js, hemos implantado la lógica necesaria para poder usar nuestra ruta de /upload. Finalmente en la interfaz nos quedaría algo así:



The screenshot shows a web interface with a dark background. At the top, there is a text input field with the placeholder text "Ingresa la letra de la canción aquí...". Below the input field is a small icon of a speech bubble with a green checkmark and a circular arrow. At the bottom of the interface, there are three buttons: "Clasificar" (green), "Borrar" (green), and "Subir Audio" (blue).



Un pequeño inconveniente de la librería `speech_recognition` es que puede dar fallos a la hora de que tan extenso sea el archivo de audio que le estemos pasando. Por lo que si es una canción larga puede llegar a fallar a la hora de la transcripción de la letra. Por otra parte es bastante sensible con el formato dado, se recomienda que se use en archivos `.wav`.

Enlace al repositorio de Git Hub

https://github.com/Paez11/Pop_detector.git

4. Conclusiones

En este proyecto de programación e inteligencia artificial, desarrollamos varios modelos para clasificar canciones pertenecientes o no al género pop a partir del análisis de sus letras. Los resultados muestran que el modelo tiene un desempeño bastante acertado, especialmente al distinguir canciones de rap clasificándolas en non-pop, lo que indica que este género presenta patrones líricos bien diferenciados respecto al pop. Sin embargo, los modelos enfrentan mayores dificultades al clasificar canciones de rock, confundiendo este género con el pop en varios casos. Esto sugiere que las letras de rock y pop comparten similitudes que el modelo no logra discriminar de manera efectiva.

Entre los modelos probados, la regresión logística y el *SGD Classifier* presentaron los mejores resultados, destacando su potencial para esta tarea. Sin embargo, los resultados también indican que una mejora significativa podría lograrse al emplear más recursos computacionales, lo que permitiría explorar arquitecturas más complejas, ajustar hiperparámetros de manera más exhaustiva y procesar un mayor volumen de datos.

También hemos aprendido cómo exportar y cargar los modelos para usarlos en aplicaciones externas a colab como una web para que muchos más usuarios puedan usarlos sin tener que acceder a colab.

En general, el análisis de letras demuestra ser una herramienta prometedora para la clasificación de géneros musicales. No obstante, para abordar mejor los casos de géneros con estilos líricos más similares, será fundamental incorporar características adicionales, como embeddings más avanzados, y explorar configuraciones algorítmicas optimizadas.