PREDICCIÓN DE GÉNERO DE PELÍCULAS

TÓPICOS AVANZADOS

ANGUIE GARCIA
MILI GALINDO
SONIA RAMÍREZ
LOURDES RODIL

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN ANALÍTICA PARA LA INTELIGENCIA DE NEGOCIOS
15 DE MARZO DE 2024

Tabla de Contenido

1 Adquisición de los datos 1
1.1 Diccionario de datos 1
1.2 Análisis Descriptivo de los datos 1
2 Limpieza del Texto 4
3 Preprocesamiento del Texto 4
3.1 Stemming 4
3.2 Lemmatization 4
4 Feature Engineering 5
4.1 Embeddings 5
4.2 Análisis de sentimientos 5
4.3 Generación de Etiquetas 6
5 Modelación 6
6 Evaluación 7
Bibliografía 8

PREDICCIÓN DE GÉNERO DE PELÍCULAS

1 Adquisición de los datos

Para este proyecto se usaron dos fuentes de datos en formato csv:

1. Datos de Entrenamiento:
   'https://github.com/sergiomora03/AdvancedTopicsAnalytics/raw/main/datasets/dataTraining.zip'
   Esta base de datos consta de 7.895 registros.

2. Datos de Prueba:
   'https://github.com/sergiomora03/AdvancedTopicsAnalytics/raw/main/datasets/dataTesting.zip'
   Esta base de datos consta de 3.383 registros.

Por lo anterior, se observa que para el proyecto se trabajará con una distribución de 70% para entrenamiento y 30% para prueba.

1.1 Diccionario de datos

La base de datos de entrenamiento cuenta con la siguiente estructura:

Nombre campo	Tipo	Descripción
Year	Integer	Corresponde al año de estreno la película
Title	Texto	Título de la Película
Plot	Texto	Resumen de trama de la película
Genres	Texto	Lista de Géneros abordados en la película
rating	Float	Calificación de la película de 1 a 10, en donde 10 es la mejor calificación dada a la película.

Tabla 1 - Diccionario de datos Entrenamiento

La base de datos de prueba cuenta con la siguiente estructura:

Nombre campo	Tipo	Descripción
Year	Integer	Corresponde al año de estreno la película
Title	Texto	Título de la Película
Plot	Texto	Resumen de trama de la película

Tabla 2 - Diccionario de datos Prueba

1.2 Análisis Descriptivo de los datos

Al observar el histograma de frecuencia de los géneros (*Figura 1*), se observa que el género más frecuente es el de drama, con 3.965 películas, seguido de comedia con 3.046 películas. Vale la pena destacar, que una película puede tener más de un género, por lo que la predicción deberá considerar esta situación.

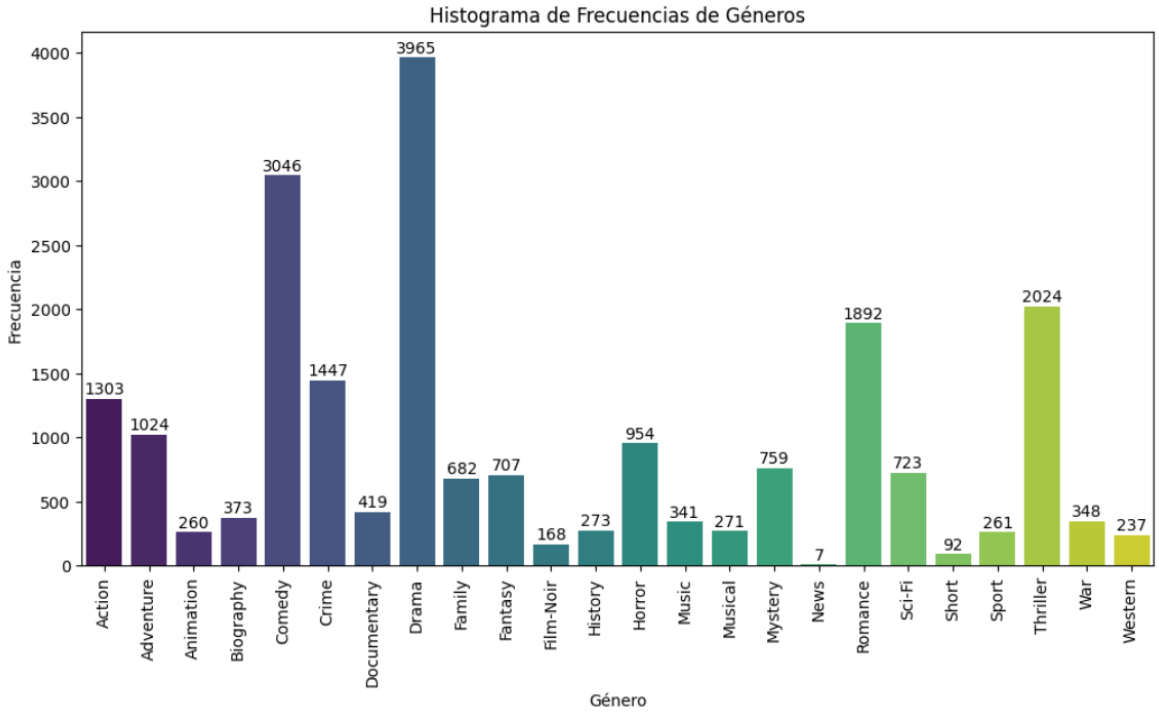

Fig 1 – Histograma de frecuencias de Géneros de Películas

Al analizar la información por año (*Figura 2*) se observa que la distribución de los datos es asimétrica hacia la derecha o positivamente sesgada, con lo cual se establece igualmente que hay mayor número de películas en los últimos 30 años.

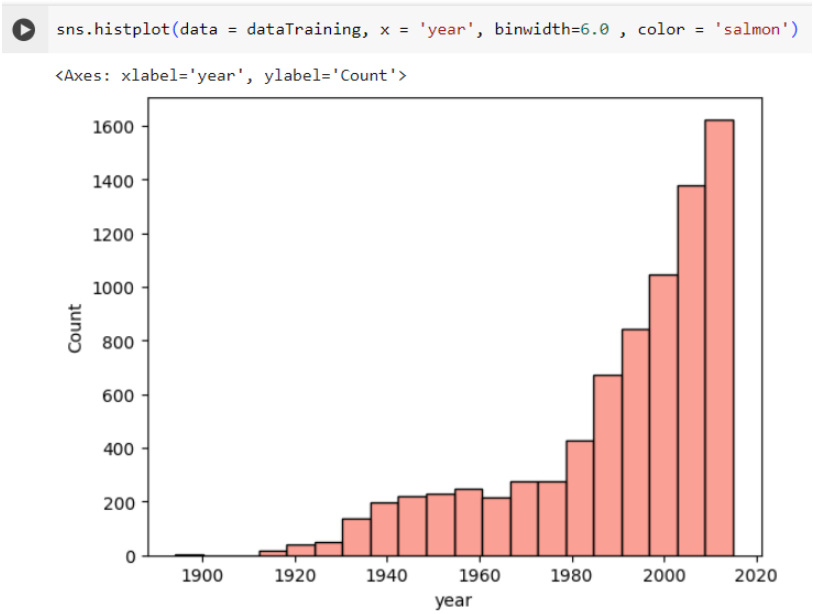

Fig 2 – Histograma de frecuencias de años en las películas

Igualmente, al evaluar la distribución del rating de las películas (*Figura 3*) se observa una concentración entre una calificación de 6 y 8, con una distribución más simétrica. No

obstante, la base de datos de prueba no cuenta con esta variable, razón por la cual será descartada para el modelo :

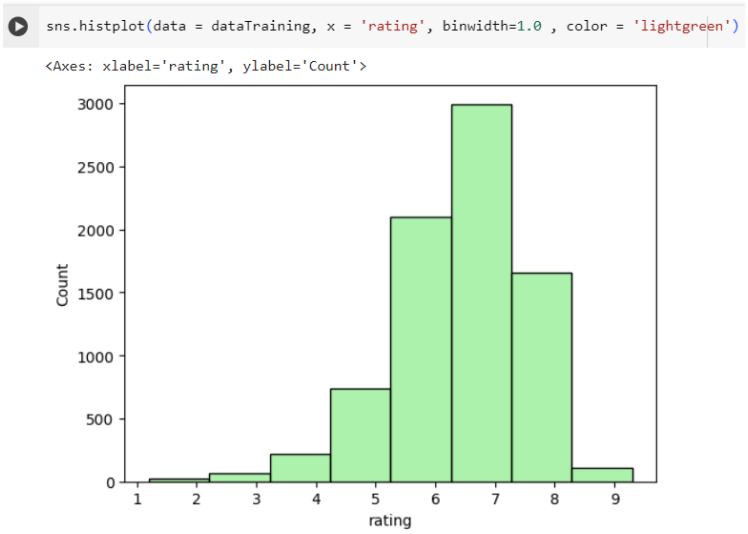

Fig 3 – Histograma de frecuencias de ratings de películas

Al analizar la matriz de correlación entre los géneros de películas (*Figura 4*) se observa que la mayor correlación positiva es entre “Familia” y “Animation” con un 0.5 y la mayor correlación negativa es entre “Triller” y “Comedy” con un -0.3. Igualmente, se observa una correlación positiva fuerte entre “Crime” y “Thriller” con 0.4:

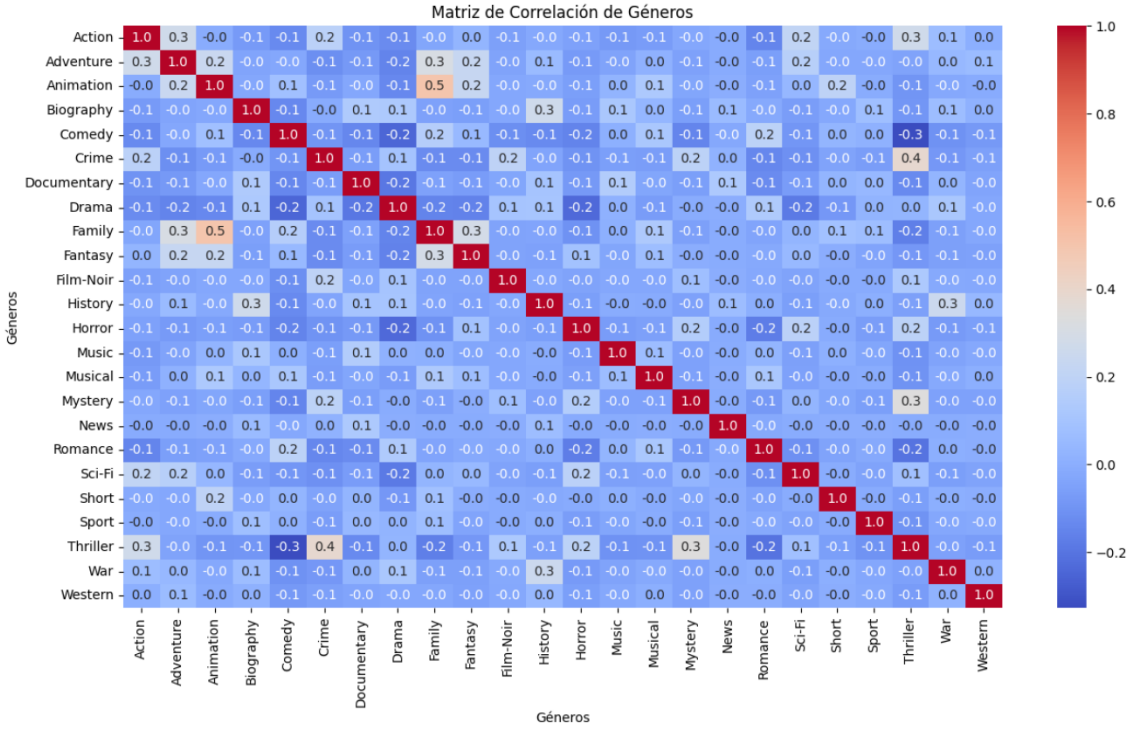

Fig 4 – Matriz de correlación de géneros de películas

2 Limpieza del Texto

Para el proyecto no fueron requeridas labores de limpieza de la base de datos, previo al paso de procesamiento, ya que no se evidenciaron datos nulos.

3 Preprocesamiento del Texto

En la etapa de preprocesamiento se aplicaron las siguientes técnicas:

3.1 Stemming

Se aplicó el algoritmo de “Snowball” (Porter, 2023) para procesar los documentos del dataset para reducir variantes morfológicas de las formas de una palabra a raíces comunes o lexemas y obtener una lista de las raíces de los documentos. Adicionalmente, se ajustan los documentos a minúsculas, se eliminan los caracteres que no sea una letra o un número, los signos de puntuación.

Se realiza un proceso de vectorización de texto utilizando la técnica TF-ID donde se ignorarán las palabras que aparecen en más del 70% de los documentos dado que al muy comunes y no aportarían mucha información.

```
stemmer = SnowballStemmer('english')
import re

def tokenize_and_stem(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9]', ' ', text)
    text = re.sub(r'[^\w\s]', '', text)
    text = ' '.join(text.split())
    tokens = [word for word in nltk.word_tokenize(text)]
    stems = [stemmer.stem(token) for token in tokens]
    return stems


vect = TfidfVectorizer(max_df=0.7,max_features=5000,stop_words='english', tokenizer=tokenize_and_stem)
X_dtm = vect.fit_transform(dataTraining['plot'])
X_dtm_title = vect.fit_transform(dataTraining['title'])
print(X_dtm.shape)
```

Fig 5 – Proceso de Stemming en código de Python

3.2 Lemmatization

Se utilizó el algoritmo de “WordNetLemmatizer” (Loper, 2023) para procesar los documentos del dataset, reducir las formas flexivas/variantes a la forma base y obtener una lista:

```
lemmatizer = WordNetLemmatizer()
# define a function that accepts text and returns a list of lemmas
def split_into_lemmas(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z0-9]', ' ', text)
    text = re.sub(r'[^\w\s]', '', text)
    words = [word for word in nltk.word_tokenize(text)]
    return [lemmatizer.lemmatize(word) for word in words]
```

```
vect = TfidfVectorizer(stop_words='english',analyzer=split_into_lemmas)
X_dtm = vect.fit_transform(dataTraining['plot'])
print(X_dtm.shape)
```

Fig 6 – Proceso de Lemmatization en código de Python

4 Feature Engineering

En la etapa de ingeniería de características se aplicaron las siguientes técnicas:

4.1 Embeddings

Se utilizó un modelo de aprendizaje profundo pre-entrenado desarrollado por Google que permite codificar oraciones en vectores de alta dimensión (Universal Sentence Encoder - USE, versión 4). Se aplica el modelo al campo “plot” en los dataset de entrenamiento y testing y las convierte en vectores de características semánticas.

```
import tensorflow as tf
import tensorflow_hub as hub
```

```
module_url = 'https://tfhub.dev/google/universal-sentence-encoder/4'
use_embed = hub.load(module_url)
```

```
train_embe_plot = use_embed(dataTraining['plot'])
test_embe_plot = use_embed(dataTesting['plot'])
```

```
train_embe_clean = use_embed(dataTraining['clean'])
test_embe_clean = use_embed(dataTesting['clean'])
```

Fig 7 – Proceso de creación de Word Embeddings en código de Python

4.2 Análisis de sentimientos

Se utilizó la biblioteca “*textblob*” para calcular la polaridad del sentimiento para cada token en training y agregar características adicionales a los datos de entrenamiento. La polaridad del sentimiento es un valor numérico que oscila entre -1 y 1, donde valores negativos indican sentimientos negativos, valores positivos indican sentimientos positivos, y 0 indica neutralidad.

```python
from textblob import TextBlob
dataTraining['sentimiento'] = dataTraining['plot'].apply(lambda texto: TextBlob(texto).sentiment.polarity)
```

Stemming y Lemmatization

```python
from scipy.sparse import hstack
text_data = dataTraining['plot']

X_years = np.array(dataTraining['year']).reshape(-1, 1)
X_rating = np.array(dataTraining['rating']).reshape(-1, 1)
X_sentimiento = np.array(dataTraining['sentimiento']).reshape(-1, 1)
X_combined = hstack([X_dtm, X_years, X_sentimiento,X_rating])
```

Fig 8 – Fracción del código Pyhton de análisis de sentimientos

4.3 Generación de Etiquetas

Se combina los embeddings con información adicional sobre las películas (año, calificación, polaridad del sentimiento) en una matriz X_combined, y transforma las etiquetas de género en una matriz binaria y_genres para su uso en el modelo de clasificación:

```python
X_combined = np.concatenate([train_embe_plot, dataTraining[['year', 'rating', 'sentimiento']].values],axis=1)
le = MultiLabelBinarizer()
y_genres = le.fit_transform(dataTraining['genres'])
```

Fig 9 – Fracción del código Pyhton para la generación de etiquetas

5 Modelación

Para el modelamiento, se tomaron dos bases, una de *train* y una de *test*. Luego, a través de la función de train_test_Split de Sckit Learn, se designó el 67% de los datos para el entrenamiento y el restante 33% se reservó para realizar las pruebas locales.

Para cada modelo se utilizaron los siguientes parámetros:

#	Nombre del modelo	Parámetros (solo los que no se usaron diferente al default)
Con Stemming		
1	Random Forest Classifier	n_jobs=-1, n_estimators=150, max_depth=15
2	XGBoost (1) (objective: binary logistic)	objective='binary:logistic
3	XGBoost (2) (OneVsRestClassifier)	Parametros por defecto n_estimators=100,learning_rate=0.1, max_depth=3
4	XGBoost (3) (OneVsRestClassifier)	n_jobs=-1, n_estimators=100,learning_rate=0.3, max_depth=5
5	Support Vector Machine	Parametros por defecto

#	Nombre del modelo	Parámetros (solo los que no se usaron diferente al default)
Con Lemmatización		
6	Random Forrest Classifier	n_jobs=-1, n_estimators=150, max_depth=15
7	XGBoost (1) (objective: binary logistic)	objective='binary:logistic'
8	XGBoost (2) (OneVsRestClassifier)	Parametros por defecto n_estimators=100,learning_rate=0.1, max_depth=3
9	XGBoost (3) (OneVsRestClassifier)	n_jobs=-1, n_estimators=100,learning_rate=0.3, max_depth=5
10	Support Vector Machine	Parametros por defecto
Con Embeddings		
11	Random Forrest Classifier	n_jobs=-1, n_estimators=150, max_depth=15
12	XGBoost (1) (objective: binary logistic)	objective='binary:logistic
13	XGBoost (2) (OneVsRestClassifier)	Parametros por defecto
14	XGBoost (3) (OneVsRestClassifier)	n_jobs=-1, n_estimators=100,learning_rate=0.3, max_depth=5
15	Support Vector Machine	Parametros por defecto
16	Red Neuronal	Dos capas relu, última capa softmax. Optimizer='adam', loss='categorical_crossentropy',

Tabla 3 – Tabla de parámetros utilizados en cada modelo de predicción

6 Evaluación

A continuación, se presenta un resumen del desempeño de los diferentes modelos desarrollados:

#	Nombre del modelo	Parámetros (solo los que no se usaron diferente al default)
Con Stemming		
1	Random Forest Classifier	0.8295
2	XGBoost (1) (objective: binary logistic)	0.8295
3	XGBoost (2) (OneVsRestClassifier)	0.8265
4	XGBoost (3) (OneVsRestClassifier)	0.8292
5	Support Vector Machine	0.5000
Con Lemmatización		
6	Random Forrest Classifier	0.8186
7	XGBoost (1) (objective: binary logistic)	0.8186
8	XGBoost (2) (OneVsRestClassifier)	0.7973
9	XGBoost (3) (OneVsRestClassifier)	0.8030
10	Support Vector Machine	0.5409
Con Embeddings		
11	Random Forrest Classifier	0.8706
12	XGBoost (1) (objective: binary logistic)	0.8706

#	Nombre del modelo	Parámetros (solo los que no se usaron diferente al default)
13	XGBoost (2) (OneVsRestClassifier)	0.9005
14	XGBoost (3) (OneVsRestClassifier)	0.8986
15	Support Vector Machine	0.5
16	Redes Neuronales	0.5

Tabla 4 – Tabla de resultados en términos de AUC para cada modelo de predicción

Para la construcción de cada uno de los modelos presentados en la Tabla 4, se evaluaron los modelos transformando los datos mediante Stemming y Lemmatization, y Word Embeddings respectivamente. Como era de esperar, los modelos en los que se utilizaron datos transformados con Tensorflow Embeddings tuvieron un mejor desempeño comparativamente con aquellos que utilizaron Stemming y Lemmatization.

Esto se debe a que utilizar Tensorflow Embeddings siempre proporciona resultados más exactos, debido a que a diferencia del Stemming y Lemmatization, no se representan las palabras como una partícula de la palabra de origen o la palabra de origen en sí, sino que al ser un "sentence encoder", el Tensorflow Embedding considera el contexto de toda la oración para generar una representación vectorial más completa.

Con respecto al desempeño de los modelos, se observa que el modelo que tuvo un mayor desempeño fue el de XGBoost utilizando los parámetros por default. La idea básica detrás de XGBoost es utilizar árboles de decisión como modelos base y luego construir un conjunto de estos árboles para mejorar la precisión de las predicciones. En el contexto de la clasificación de texto, cada árbol toma un conjunto de características extraídas del documento de texto como entrada y genera una etiqueta de clase prevista. Luego, XGBoost entrena varios árboles con los datos de entrenamiento y agrega sus predicciones para producir la predicción final para cada documento de texto.

Este modelo representa una ventaja competitiva sobre los otros modelos seleccionados ya que, a diferencia de estos, el XGBoost se entrena sobre los errores de las predicciones pasadas, y para evitar el overfitting posee diferentes técnicas de regularización integradas que lo hacen un modelo integral y difícil de superar.

Después de realizar el modelo de XGBoost para el cual se obtuvo un AUC de 0.9, se realizó un GridSearch para encontrar los mejores hiperparámetros y así poder afinar aún más los resultados del modelo.

Bibliografía

Loper, S. B. (01 de 2023). *NLTK* . Obtenido de https://www.nltk.org/_modules/nltk/stem/wordnet.html

Porter, M. (09 de 2023). Obtenido de Snowball: https://snowballstem.org/