Procesamiento de Lenguaje Natural (NLP): Análisis (clasificación) de sentimientos

Daniel Ernesto Zambrano 201914912

Importacion de Librerias

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

plt.style.use('ggplot')
import nltk
```

Leer los datos proporcionados

```
db_path = os.path.join('.', 'input_data', 'MovieReviews.csv')
df = pd.read_csv(db_path, sep = ",")
print(df.shape)
df_few = df.head(500)
print(df_few.shape)
```

```
(5000, 3)
(500, 3)
```

```
df_few.head()
```

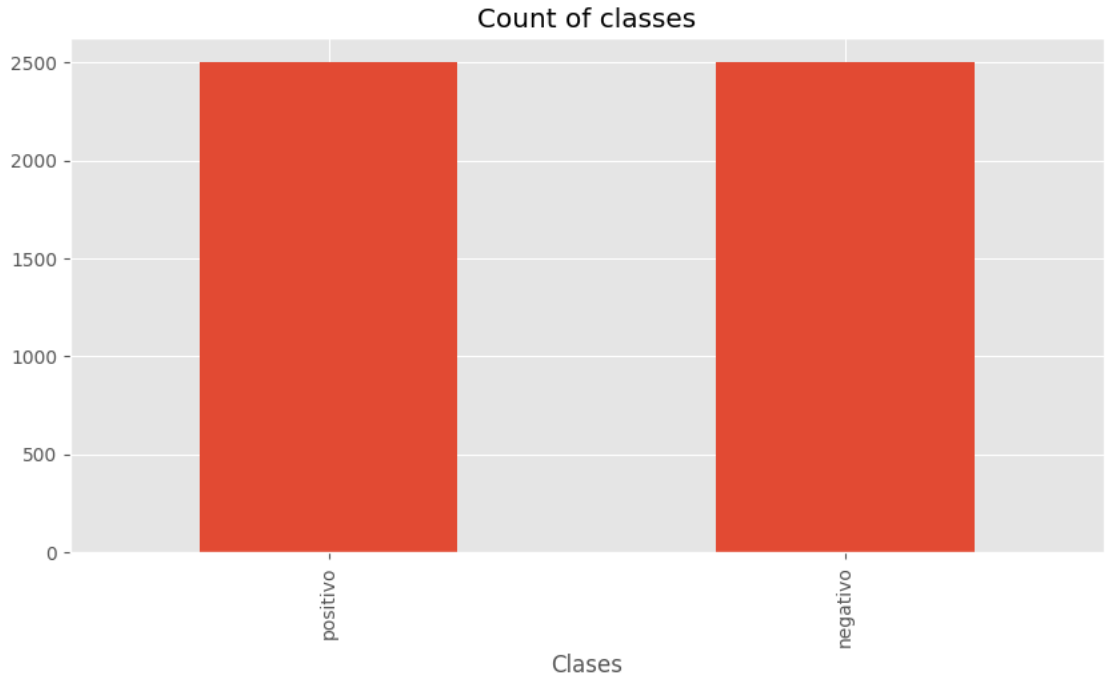
	Unnamed: 0	review_es	sentimiento
0	0	Si está buscando una película de guerra típica...	positivo
1	1	Supongo que algunos directores de películas de...	positivo
2	2	Es difícil contarle más sobre esta película si...	positivo
3	3	La película comienza muy lentamente, con el es...	positivo
4	4	Esta película es verdadera acción en su máxima...	positivo

```
df = df.rename({'Unnamed: 0': 'id'},axis=1)
df_few = df_few.rename({'Unnamed: 0': 'id'},axis=1)

df_few.head()
```

	id	review_es	sentimiento
0	0	Si está buscando una película de guerra típica...	positivo
1	1	Supongo que algunos directores de películas de...	positivo
2	2	Es difícil contarle más sobre esta película si...	positivo
3	3	La película comienza muy lentamente, con el es...	positivo
4	4	Esta película es verdadera acción en su máxima...	positivo

```
ax = df_few['sentimiento'].value_counts().plot(kind='bar',title='Count of classes',
                                   figsize=(10,5))
ax.set_xlabel('Clases')
plt.show()
```


```
ax = df['sentimiento'].value_counts().plot(kind='bar',title='Count of classes',
                                   figsize=(10,5))
ax.set_xlabel('Clases')
plt.show()

df['sentimiento'].value_counts()
```




```
positivo    2500
negativo    2500
Name: sentimiento, dtype: int64
```

Al hacer este analisis sobre el total de etiquetas para cada clase podemos ver que el problema tiene un conjunto de datos perfectamente balanceado, en particular $\frac{||Suic||}{||NonSuic||} = 1$, indicativo de que entre tamaño de las clases **no** es dispar en tamaño, luego no es necesario hacer un balanceo en etapas posteriores de procesamiento.

Con esto en consideracion, se procede a usar el archivo ya procesado por el flujo de limpieza de datos hecho en general para el proyecto

```
df_clean = pd.read_csv(os.path.join('.', 'input_data', 'MovieReviews_Clean.csv'), sep = ",")
df_clean = df_clean.rename({'Unnamed: 0': 'id'},axis=1)
df_clean.shape
```

```
(5000, 3)
```

```
df_clean.head()
```

	id	review_es	sentimiento
0	0	si está buscando una película de guerra típica...	positivo
1	1	supongo que algunos directores de películas de...	positivo
2	2	es difícil contarle más sobre esta película si...	positivo
3	3	la película comienza muy lentamente con el est...	positivo
4	4	esta película es verdadera acción en su máxima...	positivo

Procesamiento NTLK basico

```
eg = df['review_es'][167]
cl_eg = df['sentimiento'][167]
print(eg)
print(cl_eg)
```

```
Los hermanos Coen lo han vuelto a hacerlo. Tres convictos de la era de la depresión (George Clooney, John Turturro y Tim Blake Nelson) e
positivo
```

En el ejemplo puede verse que el texto refiere a un sentimiento negativo y de fastidio pero dada la etiqueta, el texto es no suicida.

```
tokens = nltk.word_tokenize(eg)
tokens[:10]
```

```
['Los', 'hermanos', 'Coen', 'lo', 'han', 'vuelto', 'a', 'hacerlo', '.', 'Tres']
```

```
tagged = nltk.pos_tag(tokens)
tagged[:10]
```

```
[('Los', 'NNP'),
 ('hermanos', 'JJ'),
 ('Coen', 'NNP'),
 ('lo', 'NN'),
 ('han', 'NN'),
 ('vuelto', 'VBZ'),
 ('a', 'DT'),
 ('hacerlo', 'NN'),
 ('.', '.'),
 ('Tres', 'VBZ')]
```

```
entities = nltk.chunk.ne_chunk(tagged)
entities.pprint()
```

```
(S
  (GPE Los/NNP)
  hermanos/JJ
  Coen/NNP
  lo/NN
  han/NN
  vuelto/VBZ
  a/DT
  hacerlo/NN
  ./.
  Tres/VBZ
  convictos/NN
  de/IN
  la/FW
  era/FW
  de/FW
  la/FW
  depresión/FW
  (/(
  (PERSON George/NNP Clooney/NNP)
  ,/,
  (PERSON John/NNP Turturro/NNP)
  y/NNP
  Tim/NNP
  (PERSON Blake/NNP Nelson/NNP)
  )/)
  escapan/VBP
  a/DT
  una/JJ
  pandilla/NN
  de/IN
```

```
cadena/FW
Mississippi/NNP
y/FW
se/FW
dirigió/FW
a/DT
la/NN
búsqueda/NN
del/NN
tesoro/NN
enterrado/NN
que/NN
financiará/JJ
sus/NN
nuevas/NN
vidas/NN
./.
En/NNP
el/NN
camino/NN
,/,
cantan/NN
en/NN
la/NN
radio/NN
y/NN
se/VBD
```

Enfoque de Analisis

El enfoque que se usara para procesar el texto estructurado que ya tenemos, es el analisis de sentimientos en las palabras o estructuras identificadas en un texto pasado por parametro.

En primera instancia se hara un analisis en base a una herramienta basada en lexico que usa diccionarios para clasificar las estructuras del texto como positivas, negativas o neutrales.

En nuestro caso, como entrada de prueba se usuara el ejemplo que ya se estructuro y se le aplicara **VADER** (Valence Aware Dictionary and sEntiment Reasoner) que es es la herramienta de análisis de sentimientos basada en reglas y léxico escogida, que está específicamente entrenada con los sentimientos expresados en las redes sociales y funciona bien en textos de otros dominios.

En segunda instancia se hara un analisis de sentimientos en base a un modelo ya entrenado usando una tarea de *masked language modeling* (MLM) que es simplemente predecir qué palabra debria llenar los espacios en blanco de una oración, recibe de entrada una mascara de texto como "la pelicula estuvo una [MASCARA]", y retorna las posibles palabras que podrian llenar tal mascara.

El modelo a usar se llama **RoBERTa** (Robustly Optimized BERT Pretraining Approach) y ees un modelo de transformadores preentrenado en un gran cuerpo de datos en multiples idiomas (incluyendo el espanol) de manera autosupervisada. **RoBERTa** está destinado principalmente a ajustarse en una tarea especifica de modelado de lenguaje, que es nuestro caso es de sentimientos.

Haremos que tambien clasifique estructuras del texto en positivas, negativas o neutras pero, en este modelo ya se consideran las relaciones entre palabras y frases, asi como tambien el sarcasmo y mas comportamientos exhibidos por las personas cuando escriben algun texto.

Paso 1: Analisis con **VADER**

```
(Ya que VADER solo analisa textos en ingles fue necesario usar google sheets para traducir todo el dataset a review en ingles, y poder vislumbar ł
```

- El resultado de analisis de VADER es una 4-tupla, que contiene 3 puntajes general para todo el texto y un puntaje compuesto que toma en cuenta los otros 3.
  - $<PostiveScore, NeutralScore, NegativeScore, CompoundScore>$.
  - El rango de los primeros 3 puntajes es $[0,1]$ siendo 0 el valor que tomaria un puntaje si el texto no denota tal sentimiento y 1 siendo el mayor valor sentimental que podria tomar el texto en un puntaje concreto.
  - El rango del puntaje compuesto es $[-1,1]$, siendo -1 un sentimiento altamente negativo, 0 un sentimiento altamente neutral y 1 un sentimiento altamente positivo.

```
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm import tqdm

sia = SentimentIntensityAnalyzer()
```

```
sia.polarity_scores("Soy un miserable y patetico pedazo de mierda")
```

```
{'neg': 0.348, 'neu': 0.652, 'pos': 0.0, 'compound': -0.4939}
```

```
sia.polarity_scores("Amigo, esto es increible")
```

```
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
```

```
sia.polarity_scores(eg)
```

```
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
```

```
import translators as ts
ts.translate_text(query_text=eg,translator="google",from_language='es', to_language='en')
```

```
'The Coen brothers have done it again. Three convicts of the era of depression (George Clooney, John Turturro and Tim Blake Nelson) escape a Mississippi chain gang and went to the search for the buried treasure that will finance their new lives. Along the way, they sing on the radio and become very sought after, as well as escapacos. Great laughs and a soundtrack that is a lesson / introduction to Bluegrass Music.Clooney is exceptional as the fast speech, use Ulysses Everett McGill quickly. Holly Hunter plays his remote wife. Turturro and Nelson are impeccable traps. Also in the cast are John Goodman and Charles Durning.dan Tyminksi provides the song of singing for George Clooney in "I am a man of constant sadness", the song of the boys in a soaked background that serves as a template for the soundtrack of Bluegrass Laden which also has Alison Krauss, Ralph Stanley, the Whites, John Hartford, the Cox and Welch Gillian family. Touching the foot of the foot, the knee with a fun for the whole family. It will be surprised with how relaxed and fun this movie is.'
```

```
df_trans_clean = pd.read_csv(os.path.join('.', 'input_data', 'Movies_Translated.csv'), sep = ",")
df_trans_clean = df_trans_clean.rename({'Unnamed: 0': 'id'},axis=1)
df_trans_clean.head()
```

	id	review_es	sentimiento	translated_text
0	0	si está buscando una película de guerra típica...	positivo	If you are looking for a typical war film, thi...
1	1	supongo que algunos directores de películas de...	positivo	I suppose that some luxury films directors wer...
2	2	es difícil contarle más sobre esta película si...	positivo	It is difficult to tell him more about this mo...
3	3	la película comienza muy lentamente con el est...	positivo	The film begins very slowly with the lifestyle...
4	4	esta película es verdadera acción en su máxima...	positivo	This film is true action at its maximum expres...

```
res = {}
puteao = []
for i, row in tqdm(df_trans_clean.iterrows(), total=len(df_trans_clean)):
    try:
        text = row['translated_text']
        df_id = row['id']
        res[df_id] = sia.polarity_scores(text)
    except AttributeError:
        puteao.append(df_id)
        print(len(puteao))
```

```
100%|██████████| 5000/5000 [00:17<00:00, 291.70it/s]
```

```
print(len(res))
```

```
5000
```

```
vaders = pd.DataFrame(res).T
vaders = vaders.reset_index().rename(columns={'index':'id'})
vaders = vaders.merge(df_clean, how='left')
```

```
vaders.head()
```

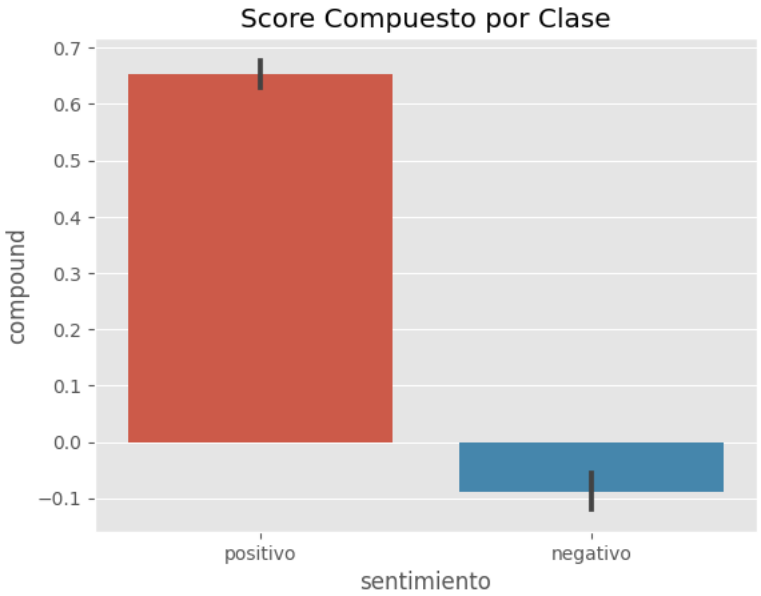
```
id    neg    neu    pos  compound                                review_es  sentimiento
```

Supuestos

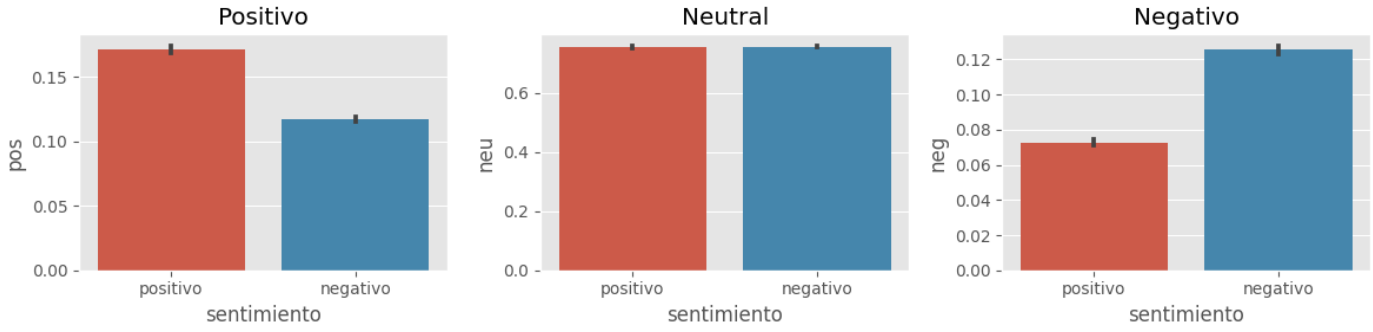
- Tomaremos de supuestos:
  - h_1: Los textos con un score compuesto negativo son textos con la etiqueta "negativo"
  - h_1^c: Los textos con un score compuesto positivo son textos con la etiqueta "positivo"
  - h_2: Los textos con un score negativo alto son textos con la etiqueta "negativo"
  - h_2^c: Los textos con un score positivo alto son textos con la etiqueta "positivo"

Para su comprobacion se hara uso de los resultados del procesamiento hecho sobre VADER, en los datos de resenas de peliculas.

```
ax = sns.barplot(data=vaders, x='sentimiento', y='compound')
ax.set_title('Score Compuesto por Clase')
plt.show()
```




```
fig, axs = plt.subplots(1, 3,figsize=(12,3))
sns.barplot(data=vaders, x='sentimiento', y='pos', ax=axs[0])
sns.barplot(data=vaders, x='sentimiento', y='neu', ax=axs[1])
sns.barplot(data=vaders, x='sentimiento', y='neg', ax=axs[2])
axs[0].set_title('Positivo')
axs[1].set_title('Neutral')
axs[2].set_title('Negativo')
plt.tight_layout()
plt.show()
```



Analisis con el modelo pre-entrenado RoBERTa

```
import torch
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from transformers import TFAutoModelForSequenceClassification
from transformers import AutoConfig
from scipy.special import softmax


MODEL = f"cardiffnlp/twitter-xlm-roberta-base-sentiment"

tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-xlm-roberta-base-sentiment")
config = AutoConfig.from_pretrained("cardiffnlp/twitter-xlm-roberta-base-sentiment")
model = AutoModelForSequenceClassification.from_pretrained("cardiffnlp/twitter-xlm-roberta-base-sentiment")
model.save_pretrained("cardiffnlp/twitter-xlm-roberta-base-sentiment")
```

```
Downloading pytorch_model.bin:   0%|          | 0.00/1.11G [00:00<?, ?B/s]
c:\Users\Usuario\Desktop\UNI\BI\B_env\lib\site-packages\huggingface_hub\file_download.py:133: UserWarning: `huggingface_hub` cache-syste
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to see activat
  warnings.warn(message)
```

```
print(eg)
sia.polarity_scores(eg)
```

```
Los hermanos Coen lo han vuelto a hacerlo. Tres convictos de la era de la depresión (George Clooney, John Turturro y Tim Blake Nelson) e
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
```

```
tokenizer.model_max_length=100000000000000


tokenizer.model_max_length
```

```
100000000000000
```

```
encoded_text = tokenizer(eg, return_tensors='pt')
output = model(encoded_text['input_ids'],encoded_text['attention_mask'])

scores = output[0][0].detach().numpy()
scores = softmax(scores)
scores_dict = {
    'roberta_neg': scores[0],

    'roberta_neu': scores[1],

    'roberta_pos': scores[2],
}

print(scores_dict)
```

```
{'roberta_neg': 0.32822815, 'roberta_neu': 0.37837037, 'roberta_pos': 0.29340148}
```

```
test= tokenizer("increible mi hermano :)", return_tensors='pt')
output = model(test['input_ids'],test['attention_mask'])
scores = output[0][0].detach().numpy()
scores = softmax(scores)
scores_dict = {
    'roberta_neg': scores[0],

    'roberta_neu': scores[1],

    'roberta_pos': scores[2],
}

print(scores_dict)
```

```
{'roberta_neg': 0.017584004, 'roberta_neu': 0.04447494, 'roberta_pos': 0.9379411}
```

```
print(len(encoded_text['input_ids'][0]))
print(len(encoded_text['attention_mask'][0]))
```

```
286
286
```

```
len(test['input_ids'][0])
```

```
8
```

```
test
```

```
{'input_ids': tensor([[     0,     23,   7612,  28236,    324, 131218,   1094,      2]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1,
1, 1]])}
```

Limitaciones de **RoBERTa**

```
(o cualquier modelo de transformadores a pequena escala, porque chatgpt ya no tiene este problema)
```

El modelo RoBERTa, solo puede procesar *textos cortos* (como los tweets), mas especificamente, textos que no generen mas de 512 tokens al realizar el preprocesamiento para estructurar el texto, y mas tecnicamente, textos que junto con los *tags* puestos por el tokenizer, [CLS] y [SEP], no generen mas de 514 tokens. Capacidad que ya se vio superada con creces por el modelo de NLP basado en transformadores GPT-3, GPT-3.5 y porsupuesto GPT-4-32k que cuenta ahora con la capacidad de evaluar textos que generen hasta **32000 tokens**

Luego es necesario considerar una manera de procesar nuestro ejemplo, y en particular, los textos a procesar dentro de los datos que sean medianamente largos o realmente largos.

Nuestra propuesta es:

1. **dividir** toda la tokenizacion en segmentos de 512 tokens, esto es $\frac{||text||}{512}$.
2. Modificar los tokens para que sean validos , es decir, añadir *tags* manualmente al tensor que retorna la tokenizacion
3. Procesar los tensores en RoBERTa.
4. Realizar un promedio ponderado de los puntajes de cada uno de los tensores.
5. Se retorna ese promedio y se almacena.

¿Por qué no en pedazos de 514?

Porque habra que añadir manualmente los tags del modelo despues de dividir la tokenizacion en segmentos de 512 tokens.

```
def sentiment_roberta(tokens):
    # get output logits from the model
    output = model(**tokens)
    # convert to probabilities
    probs = torch.nn.functional.softmax(output[0], dim=-1)
    probs = probs.detach().numpy()
    N =len(probs)

    neg,neu,pos=0,0,0

    for tensor in probs:
        neg +=tensor[0]
        neu +=tensor[1]
        pos +=tensor[2]

    probs_dict = {
            'roberta_neg': neg/N,
            'roberta_neu': neu/N,
            'roberta_pos': pos/N,
        }
    return probs_dict



chunksize = 512

input_id_chunks = list(encoded_text['input_ids'][0].split(int(chunksize - 2)))
mask_chunks = list(encoded_text['attention_mask'][0].split(int(chunksize - 2)))

for i in range(len(input_id_chunks)):
    # añadir CLS y SEP a cada chunck de tokens
    input_id_chunks[i] = torch.cat([
        torch.tensor([101]), input_id_chunks[i], torch.tensor([102])
    ])
    # añádir los attention tokens el padding de los tags
    mask_chunks[i] = torch.cat([
        torch.tensor([1]), mask_chunks[i], torch.tensor([1])
```

```
    ])
    #calcular el tamaño del padding
    pad_len = chunksize - input_id_chunks[i].shape[0]
    # revisar si la longitud del padding en el tensor es la requerida
    if pad_len > 0:
        # en caso de que no, se añade el padding necesario
        input_id_chunks[i] = torch.cat([
            input_id_chunks[i], torch.Tensor([0] * pad_len)
        ])
        mask_chunks[i] = torch.cat([
            mask_chunks[i], torch.Tensor([0] * pad_len)
        ])

for chunk in input_id_chunks:
    print(len(chunk))
# imprimir el chunk final para ver si quedo bien configurado
chunk
```

```
512
tensor([1.0100e+02, 0.0000e+00, 3.7310e+03, 1.3122e+05, 7.0000e+00, 1.3110e+03,
        3.3000e+01, 4.5900e+02, 7.7700e+02, 8.1000e+01, 2.0129e+04, 1.8800e+02,
        1.0000e+01, 6.5661e+04, 5.0000e+00, 8.4790e+04, 1.5800e+02, 2.1605e+04,
        1.9520e+03, 8.0000e+00, 2.1000e+01, 1.6150e+03, 8.0000e+00, 2.1000e+01,
        8.0000e+00, 9.3381e+04, 1.5000e+01, 2.0550e+05, 5.1053e+04, 3.6300e+03,
        5.3000e+01, 4.0000e+00, 4.9390e+03, 4.2520e+03, 9.8700e+02, 5.1600e+02,
        1.1300e+02, 1.3320e+04, 1.9550e+05, 9.9331e+04, 1.6000e+01, 1.5439e+05,
        1.9000e+01, 1.0000e+01, 2.2000e+02, 8.9174e+04, 2.2980e+03, 8.0000e+00,
        1.1503e+05, 2.2526e+05, 1.1300e+02, 4.0000e+01, 9.8086e+04, 8.4900e+02,
        1.0000e+01, 2.1000e+01, 9.7449e+04, 1.4600e+02, 2.1555e+05, 3.0957e+04,
        1.6211e+04, 4.1000e+01, 6.7225e+04, 7.0900e+02, 1.8170e+03, 4.6094e+04,
        6.1866e+04, 5.0000e+00, 3.5700e+02, 8.8000e+01, 4.6571e+04, 4.0000e+00,
        8.3100e+02, 1.0760e+03, 2.2000e+01, 2.1000e+01, 5.9770e+03, 1.1300e+02,
        4.0000e+01, 1.3430e+05, 1.9000e+01, 3.5910e+03, 1.3773e+04, 1.1400e+03,
        4.0000e+00, 4.7050e+03, 5.3300e+02, 1.5439e+05, 7.8400e+03, 5.0000e+00,
        2.2634e+04, 7.0000e+00, 6.7786e+04, 7.0000e+00, 1.1300e+02, 2.2000e+02,
        1.0247e+04, 1.4913e+05, 4.1000e+01, 1.9800e+02, 2.2000e+02, 9.5000e+01,
        1.4618e+04, 2.4800e+02, 3.9864e+04, 1.1500e+03, 1.0000e+01, 2.2928e+04,
        5.7390e+03, 7.0000e+00, 7.0000e+00, 1.3703e+04, 5.0000e+00, 4.4100e+02,
        4.7673e+04, 1.0186e+04, 1.9800e+02, 1.4708e+05, 5.3300e+02, 8.8000e+01,
        4.0071e+04, 4.1000e+01, 5.7584e+04, 4.0000e+00, 4.5270e+03, 1.7983e+05,
        3.4500e+02, 5.3800e+02, 4.3576e+04, 7.8650e+04, 4.2040e+03, 1.5758e+05,
        5.7130e+03, 5.0000e+00, 1.0938e+05, 5.3000e+01, 1.3433e+05, 2.6389e+04,
        1.0000e+01, 1.6600e+02, 1.0799e+05, 6.6200e+02, 7.0733e+04, 5.0000e+00,
        4.2520e+03, 9.8700e+02, 5.1600e+02, 1.1300e+02, 9.9331e+04, 7.7500e+02,
        1.5276e+05, 1.6200e+02, 4.8793e+04, 2.3800e+02, 1.9736e+04, 5.0000e+00,
        3.8661e+04, 2.2000e+01, 8.8000e+01, 1.3255e+05, 7.7500e+02, 4.9390e+03,
        1.8621e+04, 6.6900e+02, 1.1300e+02, 2.8166e+04, 2.4344e+04, 5.9200e+02,
        5.0000e+00, 3.4119e+04, 9.0432e+04, 7.3000e+01, 3.0430e+03, 4.6872e+04,
        2.1000e+01, 1.6553e+04, 8.0000e+00, 3.1186e+04, 1.2100e+02, 1.4787e+04,
        5.1053e+04, 3.6300e+03, 5.3000e+01, 2.2000e+01, 4.4000e+01, 8.9120e+03,
        5.3000e+01, 5.1000e+01, 4.6491e+04, 8.0000e+00, 3.4757e+04, 5.9600e+02,
        4.5320e+04, 8.3000e+02, 2.1000e+01, 8.2615e+04, 8.0000e+00, 3.8800e+02,
        1.6580e+03, 7.8400e+03, 8.0000e+00, 2.3799e+04, 3.5200e+02, 4.1592e+04,
        2.4600e+02, 4.1000e+01, 1.2966e+05, 5.3300e+02, 1.6838e+05, 1.2100e+02,
        2.1000e+01, 1.0247e+04, 1.4913e+05, 8.0000e+00, 2.2928e+04, 5.7390e+03,
        7.0000e+00, 7.0000e+00, 1.0980e+05, 4.1000e+01, 6.4160e+03, 9.4720e+03,
        1.5800e+02, 2.1320e+03, 1.6810e+03, 2.1028e+04, 2.2300e+02, 7.0000e+00,
        4.0000e+00, 1.7932e+05, 1.3811e+05, 4.0000e+00, 3.8800e+02, 1.0146e+05,
        7.0000e+00, 4.0000e+00, 4.9390e+03, 4.9307e+04, 1.3534e+04, 4.0000e+00,
        2.1000e+01, 8.6500e+03, 1.3110e+03, 4.2500e+02, 1.1300e+02, 3.0982e+04,
        2.0600e+02, 5.2700e+02, 1.6208e+05, 5.0000e+00, 7.1700e+02, 8.1914e+04,
        8.8000e+01, 8.0000e+00, 2.4600e+02, 1.4600e+02, 2.1580e+03, 4.0000e+00,
        2.1000e+01, 1.5067e+05, 1.4300e+02, 1.5800e+02, 2.2000e+02, 1.8595e+04,
        5.8000e+03, 1.2100e+02, 6.9090e+03, 2.1000e+01, 8.6500e+03, 5.0000e+00,
        5.0300e+02, 7.5309e+04, 1.3000e+01, 1.6800e+03, 1.5800e+02, 4.5900e+02,
        1.5421e+04, 1.4500e+02, 2.4600e+02, 1.1300e+02, 1.2687e+05, 4.1000e+01,
        1.9800e+02, 1.6420e+03, 5.8342e+04, 5.0000e+00, 2.0000e+00, 1.0200e+02,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
        0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
```

```
input_ids = torch.stack(input_id_chunks)
attention_mask = torch.stack(mask_chunks)

input_dict = {
    'input_ids': input_ids.long(),
```

```
    'attention_mask': attention_mask.int()
}
input_dict

 {'input_ids': tensor([[   101,      0,   3731, 131218,      7,   1311,     33,    459,    777,
             81,  20129,    188,     10,  65661,      5,  84790,    158,  21605,
           1952,      8,     21,   1615,      8,     21,      8,  93381,     15,
         205503,  51053,   3630,     53,      4,   4939,   4252,    987,    516,
            113,  13320, 195501,  99331,     16, 154388,     19,     10,    220,
          89174,   2298,      8, 115029, 225260,    113,     40,  98086,    849,
             10,     21,  97449,    146, 215552,  30957,  16211,     41,  67225,
            709,   1817,  46094,  61866,      5,    357,     88,  46571,      4,
            831,   1076,     22,     21,   5977,    113,     40, 134300,     19,
           3591,  13773,   1140,      4,   4705,    533, 154388,   7840,      5,
          22634,      7,  67786,      7,    113,    220,  10247, 149126,     41,
            198,    220,     95,  14618,    248,  39864,   1150,     10,  22928,
           5739,      7,      7,  13703,      5,    441,  47673,  10186,    198,
         147078,    533,     88,  40071,     41,  57584,      4,   4527, 179833,
            345,    538,  43576,  78650,   4204, 157580,   5713,      5, 109383,
             53, 134327,  26389,     10,    166, 107987,    662,  70733,      5,
           4252,    987,    516,    113,  99331,    775, 152755,    162,  48793,
            238,  19736,      5,  38661,     22,     88, 132547,    775,   4939,
          18621,    669,    113,  28166,  24344,    592,      5,  34119,  90432,
             73,   3043,  46872,     21,  16553,      8,  31186,    121,  14787,
          51053,   3630,     53,     22,     44,   8912,     53,     51,  46491,
              8,  34757,    596,  45320,    830,     21,  82615,      8,    388,
           1658,   7840,      8,  23799,    352,  41592,    246,     41, 129661,
            533, 168378,    121,     21,  10247, 149126,      8,  22928,   5739,
              7,      7, 109798,     41,   6416,   9472,    158,   2132,   1681,
          21028,    223,      7,      4, 179322, 138109,      4,    388, 101455,
              7,      4,   4939,  49307,  13534,      4,     21,   8650,   1311,
            425,    113,  30982,    206,    527, 162076,      5,    717,  81914,
             88,      8,    246,    146,   2158,      4,     21, 150674,    143,
            158,    220,  18595,   5800,    121,   6909,     21,   8650,      5,
            503,  75309,     13,   1680,    158,    459,  15421,    145,    246,
            113, 126874,     41,    198,   1642,  58342,      5,      2,    102,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0,      0,
              0,      0,      0,      0,      0,      0,      0,      0]]),
  'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

sentiment_roberta(input_dict)

 {'roberta_neg': 0.2971102297306061,
  'roberta_neu': 0.36257898807525635,
  'roberta_pos': 0.3403107821941376}


def chunkify_tokens(ptokens):

    chunksize = 512

    input_id_chunks = list(ptokens['input_ids'][0].split(int(chunksize - 2)))
    mask_chunks = list(ptokens['attention_mask'][0].split(int(chunksize - 2)))

    for i in range(len(input_id_chunks)):
        # añadir CLS y SEP a cada chunck de tokens
        input_id_chunks[i] = torch.cat([
            torch.tensor([101]), input_id_chunks[i], torch.tensor([102])
        ])
```

```
        # añádir los attention tokens el padding de los tags
        mask_chunks[i] = torch.cat([
            torch.tensor([1]), mask_chunks[i], torch.tensor([1])
        ])
        #calcular el tamaño del padding
        pad_len = chunksize - input_id_chunks[i].shape[0]
        # revisar si la longitud del padding en el tensor es la requerida
        if pad_len > 0:
            # en caso de que no, se añade el padding necesario
            input_id_chunks[i] = torch.cat([
                input_id_chunks[i], torch.Tensor([0] * pad_len)
            ])
            mask_chunks[i] = torch.cat([
                mask_chunks[i], torch.Tensor([0] * pad_len)
            ])
        input_ids = torch.stack(input_id_chunks)
        attention_mask = torch.stack(mask_chunks)

        input_dict = {
            'input_ids': input_ids.long(),
            'attention_mask': attention_mask.int()
            }
        return input_dict



def polarity_scores_roberta(text):
    tokens = tokenizer(text, return_tensors='pt')
    if len(tokens[0]) > 512:
        input_dict = chunkify_tokens(tokens)
        r_dict = sentiment_roberta(input_dict)
    else:
        r_dict = sentiment_roberta(tokens)
    return r_dict


res = {}
puteaoRuntime = []
for i, row in tqdm(df_clean.iterrows(), total=len(df_clean)):
    try:
        text = row['review_es']
        myid = row['id']
        vader_result = sia.polarity_scores(text)
        vader_result_rename = {}
        for key, value in vader_result.items():
            vader_result_rename[f"vader_{key}"] = value
        roberta_result = polarity_scores_roberta(text)
        both = {**vader_result_rename, **roberta_result}
        res[myid] = both
    except RuntimeError:
        puteaoRuntime.append(myid)
        print(f'puteao en Runtime {len(puteaoRuntime)}')
    except AttributeError:
        puteao.append(myid)
        print(f'puteao {len(puteao)}')
```

```
  0%|          | 9/5000 [00:04<50:46,  1.64it/s]puteao en Runtime 1
puteao en Runtime 2
  0%|          | 16/5000 [00:08<48:04,  1.73it/s]puteao en Runtime 3
  1%|          | 34/5000 [00:17<37:30,  2.21it/s]puteao en Runtime 4
puteao en Runtime 5
  1%|          | 39/5000 [00:18<21:55,  3.77it/s]puteao en Runtime 6
  1%|          | 46/5000 [00:21<30:12,  2.73it/s]puteao en Runtime 7
  1%|▏         | 66/5000 [00:29<42:02,  1.96it/s]puteao en Runtime 8
  2%|▏         | 80/5000 [00:33<17:39,  4.64it/s]puteao en Runtime 9
  2%|▏         | 83/5000 [00:36<46:54,  1.75it/s]puteao en Runtime 10
  2%|▏         | 89/5000 [00:38<48:37,  1.68it/s]puteao en Runtime 11
  2%|▏         | 113/5000 [00:50<39:27,  2.06it/s] puteao en Runtime 12
  3%|▎         | 127/5000 [00:57<46:40,  1.74it/s]puteao en Runtime 13
  3%|▎         | 129/5000 [00:57<32:54,  2.47it/s]puteao en Runtime 14
  3%|▎         | 143/5000 [01:03<35:26,  2.28it/s]puteao en Runtime 15
puteao en Runtime 16
  3%|▎         | 149/5000 [01:04<23:03,  3.51it/s]puteao en Runtime 17
  3%|▎         | 153/5000 [01:05<24:38,  3.28it/s]puteao en Runtime 18
  3%|▎         | 155/5000 [01:05<19:46,  4.08it/s]puteao en Runtime 19
  3%|▎         | 160/5000 [01:07<21:00,  3.84it/s]puteao en Runtime 20
puteao en Runtime 21
```

```
  3%|▌          | 168/5000 [01:09<29:25,  2.74it/s]puteao en Runtime 22
  4%|▌          | 175/5000 [01:12<28:30,  2.82it/s]puteao en Runtime 23
  4%|▌          | 179/5000 [01:15<45:07,  1.78it/s]puteao en Runtime 24
  4%|▌          | 188/5000 [01:17<20:53,  3.84it/s]puteao en Runtime 25
  4%|▌          | 190/5000 [01:18<18:57,  4.23it/s]puteao en Runtime 26
  4%|▌          | 217/5000 [01:29<30:14,  2.64it/s]puteao en Runtime 27
  5%|▌          | 226/5000 [01:32<22:25,  3.55it/s]puteao en Runtime 28
  5%|▌          | 238/5000 [01:36<32:23,  2.45it/s]puteao en Runtime 29
  5%|▌          | 240/5000 [01:37<28:57,  2.74it/s]puteao en Runtime 30
  5%|▌          | 248/5000 [01:39<33:12,  2.38it/s]puteao en Runtime 31
  5%|▌         | 250/5000 [01:40<25:32,  3.10it/s]puteao en Runtime 32
  6%|▌          | 275/5000 [01:50<43:44,  1.80it/s]puteao en Runtime 33
  6%|▌          | 280/5000 [01:51<28:19,  2.78it/s]puteao en Runtime 34
puteao en Runtime 35
  6%|▌          | 301/5000 [01:58<38:36,  2.03it/s]puteao en Runtime 36
  6%|▌          | 307/5000 [02:00<18:19,  4.27it/s]puteao en Runtime 37
  6%|▌          | 309/5000 [02:01<41:23,  1.89it/s]puteao en Runtime 38
  6%|▋          | 315/5000 [02:03<32:36,  2.39it/s]puteao en Runtime 39
  6%|▋          | 318/5000 [02:04<29:01,  2.69it/s]puteao en Runtime 40
  6%|▋          | 323/5000 [02:06<32:38,  2.39it/s]puteao en Runtime 41
  7%|▋          | 341/5000 [02:14<21:08,  3.67it/s]  puteao en Runtime 42
puteao en Runtime 43
puteao en Runtime 44
puteao en Runtime 45
  7%|▋          | 360/5000 [02:20<35:05,  2.20it/s]puteao en Runtime 46
  7%|▋          | 362/5000 [02:21<26:22,  2.93it/s]puteao en Runtime 47
  7%|▋          | 366/5000 [02:22<25:17,  3.05it/s]puteao en Runtime 48
puteao en Runtime 49
puteao en Runtime 50
  8%|▊          | 377/5000 [02:26<42:17,  1.82it/s]puteao en Runtime 51
  8%|▊          | 383/5000 [02:29<39:08,  1.97it/s]puteao en Runtime 52
  8%|▊          | 386/5000 [02:30<25:24,  3.03it/s]puteao en Runtime 53
  8%|▊          | 396/5000 [02:33<29:29,  2.60it/s]puteao en Runtime 54
  8%|▊          | 401/5000 [02:35<28:50,  2.66it/s]puteao en Runtime 55
  8%|▊          | 407/5000 [02:36<18:08,  4.22it/s]puteao en Runtime 56
puteao en Runtime 57
```

```
err = df_clean['review_es'][662]
```

```
print(err)
```

```
uno de los menos alabados del reciente período de asuntos asiáticos los asuntos de acción gojoe es una oferta más lenta y a menudo curio
```

```
tokens = tokenizer(err, return_tensors='pt')
print(len(tokens[0]))
print(tokens)
```

```
763
{'input_ids': tensor([[     0,   3269,      8,    388,   5005,      6,  87858,   1140,    146,
          97425,  23901,      8,  38363,      7,   5644,  80144,    388,  38363,
              7,      8,  38614,    738,    513,     13,    198,    220,  19806,
           1005, 127881,    113,     10,   8026,    246, 152694,   1788,     22,
           4537,  36018,   2733,    158,  65066,      7,    113, 206291,      7,
          43250,  34757,  19187,     41,    110,   2168, 194978,   5799,      8,
             21, 197866,  13245,   1035,     36,     21,   4816, 222439,    113,
             21,  45609,   7074,     41, 148690,     10,   1642,  58342,    196,
            459,     41,   1817, 105324,      7,   1005, 129959,      7,     40,
          54312,     19,     22,  11029,   1788,  27749,    880,  91018,  43701,
          20605,  13124,  58453,  44905,    220,  12433,     11, 159556,    220,
          61785,      8,   1207,   1073,   3290,  20394,     22,     51,     51,
          35062,   1005,  15832,  24725,     34,   6781,     21,   4816,    198,
              8,   1585,    350,     14,     51,   2667,    236,  53953,    516,
            113,    569,  14570,  77793,    846,     41,  13773,     21, 219783,
          47894,     53,   4017,    144,  77793,    846,    146, 208947,    738,
            236,     13,     88, 237132,  78674,     11,     31,   9204,   9850,
             51,  13773,   1846,  67573,  20034,      8,    166,  28340, 176243,
          14597,   1788,   1642,   2366,     88,   4681,      8,    388,  77793,
           5790,   4705,     21,  58342,     40, 160891,     22,    220,  60853,
              8,    211,    223,  55105,    113,     22,     88,   2941,      8,
           1585,    350,     14,     21,  19538,    318,  23346,     51, 134465,
             22,     88,     41,     88,  18696,   2773,    198,     88,  91018,
         209267,    151,     41,   1585,    350,     14, 192836,     10,     51,
          27219,    158,    166, 173476, 101612,    113,    121,     41,     88,
         237132,  78674,     11,     31,     40,     22,  12176,   1479,     10,
             21, 101612,      8,    459,     41,   9204,  13773, 196310,   1585,
            350,     14,   1570,  37419,   1005,  99662,   8972,  25490,   9877,
            196,     88,  18095,      8,     41,    166, 101612,    362,  27315,
            459,   9859,      6, 226509,     22,  25906,   1010,     56,    246,
            158,     88,   3307,  27749,   6604,    110,   1570,     22, 134465,
         118754,  11918,     40,  46194,   7337,     88,  14364,    221,    519,
```

```
     6,  12787,     14,     40,  22311,    145,    473,    533,    220,
   393,  51781,    158,  55185,    128,   3911,     88,  11544,      8,
    21,  75479,  11928,  51849,    113,    158, 135588,    880,    224,
   318,   6000,     42,    880,    224,   1010,     14,   1506,     21,
 69809,      8,   1005,  36843,   1736,   3617,    121,     41,     21,
154925,     40,  30574,  18298,   6416,   2054,     51,   2855,   5388,
     8, 219783,    113,    269,   7119,    121,   1011,     51,     82,
  8077,     31,     99,  28237,      7,  45271,   2054,     51,  39813,
     8,   1207,   1073,   3290,     10,   2855,   1235,      8,     21,
 58342,   1788,  28518,      8,    388,  36843, 131669,   1183,   1028,
 84832,     90,    198,    220,   1207,   1073,   3290,      8, 167815,
   113,    166,   1883,    158,   1005,  23901,     41,    377,  62733,
  1005,     41,    388, 170745,      7,      8,   1207,   1073,   3290,
    21, 157824,   3290,      8,  17719,  15894, 119267,    372,    198,
  4247,   9877,   9863,  28462,    113, 103537,     85,     51,  38363,
 23306, 184357,   1846,  84365,   8864, 115940,   5535,     34,    198,
  1466,    157,    113, 159583,    158,     51,  79702,    991,  32037,
   170,   3119,    533,   1585,    350,     14,  36556,     41,  29303,
   157,    978,     10,  70699,   5904,    220,    187,  33003,  13700,
102494,     11,  46518,   1683,    533,     88, 237132,  78674,     11,
    31,      8,    388,  17013,  29263,     90,   3262,    188,   3767,
  4197,   6991,  45606,    576,  19178,    533,     51,  46491,  23335,
```

```
results_df = pd.DataFrame(res).T
results_df = results_df.reset_index().rename(columns={'index': 'id'})
results_df = results_df.merge(df, how='left')
```

```
df_clean.head()
```

	id	review_es	sentimiento
0	0	si está buscando una película de guerra típica...	positivo
1	1	supongo que algunos directores de películas de...	positivo
2	2	es difícil contarle más sobre esta película si...	positivo
3	3	la película comienza muy lentamente con el est...	positivo
4	4	esta película es verdadera acción en su máxima...	positivo

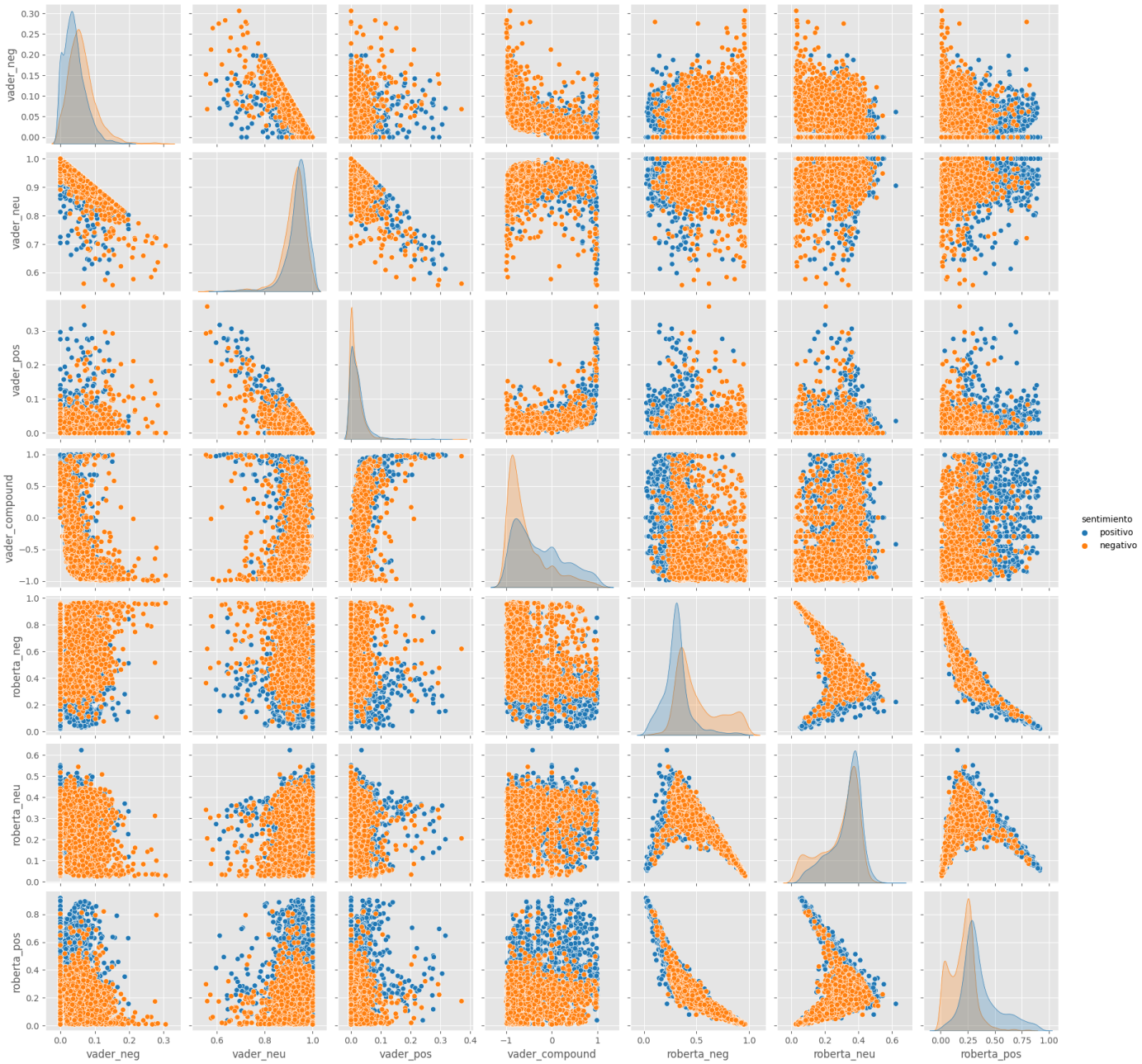
```
results_df.columns
```

```
Index(['id', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
       'roberta_neg', 'roberta_neu', 'roberta_pos', 'review_es',
       'sentimiento'],
      dtype='object')
```

```
results_df.head()
```

	id	vader_neg	vader_neu	vader_pos	vader_compound	roberta_neg	roberta_neu	roberta_pos	review_es	sentimiento
0	0	0.097	0.891	0.012	-0.9287	0.508365	0.356195	0.135440	Si está buscando una película de guerra típica...	positivo
1	1	0.047	0.903	0.050	0.4019	0.280555	0.396531	0.322914	Supongo que algunos directores de películas de...	positivo
2	2	0.085	0.897	0.018	-0.9042	0.488104	0.306203	0.205693	Es difícil contarle más sobre esta película si...	positivo
									La película comienza muy	

```
sns.pairplot(data=results_df,
             vars=['vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
                   'roberta_neg', 'roberta_neu', 'roberta_pos'],
             hue='sentimiento',
             palette='tab10')
plt.show()
```


vader_neg
vader_neu
vader_pos
vader_compound
roberta_neg
roberta_neu
roberta_pos
sentimiento
positivo
negativo

```
results_df.to_csv('roberta_vader_results_MOVIES.csv', index=True)
```

```
groundtruth = df_clean['sentimiento']
groundtruth = groundtruth.replace('positivo', 1.0)
groundtruth = groundtruth.replace('negativo', 0.0)
groundtruth.head()
```

```
0    1.0
1    1.0
2    1.0
3    1.0
4    1.0
Name: sentimiento, dtype: float64
```

```
def classify_sentiment_VADER(row):
    """Classifies sentiment of a text based on VADER sentiment analysis results.

    Args:
        row (pandas.Series): A row of a pandas DataFrame containing VADER sentiment analysis results.

    Returns:
        Tuple[str, float]: A tuple containing the sentiment label and score.

    """
    if row['compound'] > 0.05:
        return [1, row['compound']]
    elif row['compound'] < -0.05:
        return [0, row['compound']]
    else:
        if row['neg'] > row['pos']:
            return [0, row['neg']]
        elif row['neg'] < row['pos']:
            return [1, row['pos']]
        else:
            return [0, row['neu']]

results_vader_final = vaders.apply(classify_sentiment_VADER, axis=1, result_type='expand')
results_vader_final.columns = ['Sentiment', 'Score']
```

```
results_vader_final.head()
```

	Sentiment	Score
0	1.0	0.8597
1	1.0	0.9842
2	1.0	0.8715
3	0.0	-0.8748
4	1.0	0.9533

```
results_vader_final['Sentiment'].value_counts().plot(kind='bar')
```

```
<Axes: >
```

```
results_vader_final.shape
```

```
(5000, 2)
```

```
from sklearn.metrics import accuracy_score, classification_report

print(f"El score de precision del analisis de VADER sobre el dataset fue: {accuracy_score(results_vader_final['Sentiment'], groundtruth):.2f}
print(classification_report(results_vader_final['Sentiment'], groundtruth))
```

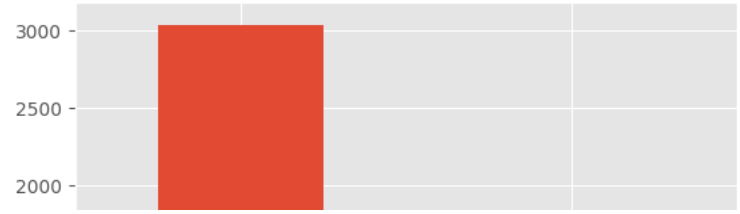
```
El score de precision del analisis de VADER sobre el dataset fue: 0.70
              precision    recall  f1-score   support

         0.0       0.55      0.79      0.64      1734
         1.0       0.85      0.65      0.74      3266

    accuracy                           0.70      5000
   macro avg       0.70      0.72      0.69      5000
weighted avg       0.75      0.70      0.71      5000
```

```
def classify_sentiment_ROBERTA(row):
    """Classifies sentiment of a text based on ROBERTA sentiment analysis results.

    Args:
        row (pandas.Series): A row of a pandas DataFrame containing ROBERTA sentiment analysis results.

    Returns:
        Tuple[str, float]: A tuple containing the sentiment label and score.

    """
    if row['roberta_neg'] > row['roberta_pos']:
        return [0, row['roberta_neg']]
    elif row['roberta_neg'] < row['roberta_pos']:
        return [1, row['roberta_pos']]
    else:
        return [0, row['roberta_neu']]


results_roberta_final = results_df.apply(classify_sentiment_ROBERTA, axis=1, result_type='expand')
results_roberta_final.columns = ['Sentiment', 'Score']


results_roberta_final.head()
```

	Sentiment	Score
0	0.0	0.508365
1	1.0	0.322914
2	0.0	0.488104
3	0.0	0.338856
4	0.0	0.386783

```
results_roberta_final['Sentiment'].value_counts().plot(kind='bar')
```

```
<Axes: >
```




```
results_roberta_final.shape
```

```
(4338, 2)
```

```
roberta_metrics = groundtruth.drop(groundtruth.index[puteao+puteaoRuntime])
roberta_metrics.shape
```

```
(4338,)
```

```
print(f"El score de precision del analisis de ROBERTA sobre el dataset fue: {accuracy_score(results_roberta_final['Sentiment'], roberta_metri
print(classification_report(results_roberta_final['Sentiment'], roberta_metrics))
```

```
El score de precision del analisis de ROBERTA sobre el dataset fue: 0.72
              precision    recall  f1-score   support

         0.0       0.91      0.66      0.77      3036
         1.0       0.52      0.85      0.65      1302

    accuracy                           0.72      4338
   macro avg       0.72      0.76      0.71      4338
weighted avg       0.80      0.72      0.73      4338
```