

Descargo el dataset con las reviews en la categoría Pet supplies

```
!pip install wordcloud
```

```
Collecting wordcloud
  Downloading wordcloud-1.9.3-cp39-cp39-win_amd64.whl (300 kB)
----- 300.6/300.6 kB 2.3 MB/s eta 0:00:00
Requirement already satisfied: matplotlib in c:\users\drakharys\anaconda3\lib\site-packages (from wordcloud) (3.5.2)
Requirement already satisfied: numpy>=1.6.1 in c:\users\drakharys\anaconda3\lib\site-packages (from wordcloud) (1.21.5)
Requirement already satisfied: pillow in c:\users\drakharys\anaconda3\lib\site-packages (from wordcloud) (9.2.0)
Requirement already satisfied: packaging>=20.0 in c:\users\drakharys\anaconda3\lib\site-packages (from matplotlib->wordcloud) (21.3)
Requirement already satisfied: cycler>=0.10 in c:\users\drakharys\anaconda3\lib\site-packages (from matplotlib->wordcloud) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\drakharys\anaconda3\lib\site-packages (from matplotlib->wordcloud) (4.22.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\drakharys\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.4.5)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\drakharys\anaconda3\lib\site-packages (from matplotlib->wordcloud) (2.8.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\drakharys\anaconda3\lib\site-packages (from matplotlib->wordcloud) (3.0.9)
Requirement already satisfied: six>=1.5 in c:\users\drakharys\anaconda3\lib\site-packages (from python-dateutil->matplotlib->wordcloud) (1.16.0)
Installing collected packages: wordcloud
Successfully installed wordcloud-1.9.3
```

```
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')
from nltk.probability import FreqDist
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from gensim.models import Word2Vec
from sklearn.decomposition import PCA

def calcular_cardinalidad_vocabulario(texto):
    palabras = word_tokenize(texto.lower())

def obtener_ngrams(texto, n=2, num=10):
    palabras = word_tokenize(texto.lower())
    n_grams = [tuple(palabras[i:i+n]) for i in range(len(palabras)-n+1)]
    frecuencia = FreqDist(n_grams)
    return frecuencia.most_common(num)

def analisis_sentimiento_basado_en_reglas(texto, sentimiento1, sentimiento2):

    # Para contar la frecuencia de palabras clave en el texto
    frecuencia_positivas = sum(texto.lower().count(palabra) for palabra in sentimiento1)
    frecuencia_negativas = sum(texto.lower().count(palabra) for palabra in sentimiento2)

    # Calcula el puntaje de sentimiento
    puntaje_sentimiento = frecuencia_positivas - frecuencia_negativas

    # Mis umbrales de clasificación
    umbral_positivo = 1
    umbral_negativo = -1

    # Clasificar la revisión
    if puntaje_sentimiento > umbral_positivo:
        return "Positivo"
    elif puntaje_sentimiento < umbral_negativo:
        return "Negativo"
    else:
        return "Neutral"

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!

# Esta casilla solo ejecutar si es en Colab
!wget https://datarepo.eng.ucsd.edu/mcauley_group/data/amazon_v2/categoryFilesSmall/Pet_Supplies_5.json.gz

--2024-04-07 13:44:42-- https://datarepo.eng.ucsd.edu/mcauley_group/data/amazon_v2/categoryFilesSmall/Pet_Supplies_5.json.gz
Resolving datarepo.eng.ucsd.edu (datarepo.eng.ucsd.edu)... 132.239.8.30
Connecting to datarepo.eng.ucsd.edu (datarepo.eng.ucsd.edu)[132.239.8.30]:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 306131006 (292M) [application/x-gzip]
Saving to: 'Pet_Supplies_5.json.gz'

Pet_Supplies_5.json 100%[=====>] 291.95M 21.7MB/s in 12s

2024-04-07 13:45:03 (23.7 MB/s) - 'Pet_Supplies_5.json.gz' saved [306131006/306131006]
```

```
datos = pd.read_json("Pet_Supplies_5.json.gz", lines=True, chunksize=50000)
datos = pd.concat(datos)
print(datos.shape)
datos.head()
```

```
(2098325, 12)
```

	overall	vote	verified	reviewTime	reviewerID	asin	reviewerName
0	3	2	True	12 2, 2016	A2KN4FJVI2TZSF	0972585419	M.G.

```
1 2 NaN True 10 17 2016 A2DMA4DD66 IDBV 0072585419 Susan B
```

```
variables_seleccionadas = ["overall", "reviewerName", "reviewText", "summary"]
datos = datos.sample(n=50000, random_state=42)
datos = datos[variables_seleccionadas]
datos.dropna(subset=['reviewText'], inplace=True)
print(datos.shape)
datos.head()
```

```
(49988, 4)
```

	overall	reviewerName	reviewText	summary
1055801	5	Noel	The large harness on its smallest settings fit...	EzyDog makes the best dog harness
2080252	5	Laurie L. Jarvis	received	Five Stars
1814786	5	Pamela U. Moore	This is the only product like this, that my ol...	The only one that helps
1167559	5	Cathy H.	I needed a carrier for a stray cat I've been f...	Perfect carrier.

Haz doble clic (o pulsa Intro) para editar

```
datos['cardinalidad_vocabulario'] = datos['reviewText'].apply(calcular_cardinalidad_vocabulario)
datos.head()
```

	overall	reviewerName	reviewText	summary	cardinalidad_vocabulario
1055801	5	Noel	The large harness on its smallest settings fit...	EzyDog makes the best dog harness	78
2080252	5	Laurie L. Jarvis	received	Five Stars	1
		Pamela U.	This is the only product like	The only	

```
# Calculo la media de la cardinalidad
media_cardinalidad = round(datos['cardinalidad_vocabulario'].mean(),2)

print("La media de la cardinalidad del vocabulario es:", media_cardinalidad)
```

La media de la cardinalidad del vocabulario es: 37.56


```
distribucion_estrellas = datos['overall'].value_counts().sort_index()
print("Distribución de reviews por número de estrellas:")
print(distribucion_estrellas)
```

Distribución de reviews por número de estrellas:

overall	
1	3273
2	2528
3	4273
4	6941
5	32973

Name: count, dtype: int64

```
num_positivas = datos[datos['overall'] >= 4].shape[0] # Tomo que 4 y 5 estrellas son positivas y 1 ,2 son más bien negativas
num_negativas = datos[datos['overall'] <= 2].shape[0]
print("Número de reviews positivas:", num_positivas)
print("Número de reviews negativas:", num_negativas)
```

 Número de reviews positivas: 39914
Número de reviews negativas: 5801

```
texto completo = ' '.join(datos['reviewText'])
```

```
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



```
# Tokenizo
tokenized_reviews = [word_tokenize(review.lower()) for review in datos['reviewText']]

# Entreno el modelo Word2Vec
model = Word2Vec(tokenized_reviews, vector_size=100, window=5, min_count=1, workers=4)

palabras_interes = ['good', 'bad', 'quality', 'price', 'service']

# Obtengo las palabras más similares a cada palabra de interés
palabras_similares = {}
for palabra in palabras_interes:
    similares = model.wv.most_similar(palabra, topn=10)
    palabras_similares[palabra] = similares

# Obtengo los embeddings de las palabras de interés y sus similares
palabras_embeddings = {}
for palabra in palabras_interes:
    similares = [palabra] + [sim[0] for sim in palabras_similares[palabra]]
    embeddings = [model.wv[p] for p in similares]
    palabras_embeddings[palabra] = embeddings

# aplico PCA
pca = PCA(n_components=2)
palabras_embeddings_pca = {palabra: pca.fit_transform(embeddings) for palabra, embeddings in palabras_embeddings.items()}

plt.figure(figsize=(10, 8))
for palabra, embeddings in palabras_embeddings_pca.items():
    plt.scatter(embeddings[:, 0], embeddings[:, 1], label=palabra)
    for i, (x, y) in enumerate(embeddings):
        # Verifico que haya suficientes palabras similares disponibles
        if i < len(palabras_similares[palabra]):
            plt.annotate(palabras_similares[palabra][i][0], (x, y), textcoords="offset points", xytext=(0,10), ha='center')
plt.xlabel('Componente principal 1')
plt.ylabel('Componente principal 2')
plt.title('Visualización de embeddings de palabras y sus similares')
plt.legend()
plt.show()
```



```
frecuencia_sentimientos = datos["sentimiento"].value_counts()

print("Número de registros de cada sentimiento:")
print(frecuencia_sentimientos)

Número de registros de cada sentimiento:
sentimiento
Neutral      46602
Positivo     3293
Negativo       93
Name: count, dtype: int64
```

Conclusiones a la exploración

- Por un lado veo que la media de cardinalidad de las reviews es de 37, 5 palabras únicas
- Que la distribución de las estrellas me indican que hay más comentarios positivos que negativos y esto se vuelve a constatar con el número de reviews positivas y negativas
- Con lo n-grams me doy cuenta de que voy a tener que preprocesar los datos, ya que de momentos son las conjunciones que se repiten más a menudo junto con los espacios
- En la nube de palabras queda claro que la categoría del dataset va sobre mascotas
- Con el Word2Vec veo que la asociación de "seller" y "value" con "price" y "quality", sugiere conexiones semánticas entre estas palabras. La relación de "terrible" con "bad" y la asociación de palabras como "wonderful", "great" y "high" con "good" indican la polaridad de las que tienen las reviews
- Por último creé una función para valorar el sentimiento general de la review y me doy cuenta que la mayoría dan sentimientos neutros

✓ Preprocesado

```
import string
import nltk
import pandas as pd
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

def preprocesar_reviews(review):
    review = eliminar_puntuacion(review)
    review = convertir_minusculas(review)
    review = eliminar_stopwords(review)
    review = stem_palabras(review)
    return review

def eliminar_puntuacion(texto):
    translator = str.maketrans('', '', string.punctuation)
    return texto.translate(translator)

def convertir_minusculas(texto):
    return texto.lower()

def eliminar_stopwords(texto):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(texto)
    words = [word for word in words if word not in stop_words]
    return ' '.join(words)

def stem_palabras(texto):
    ps = PorterStemmer()
    words = word_tokenize(texto)
    words = [ps.stem(word) for word in words]
    return ' '.join(words)

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data] Package stopwords is already up-to-date!

review = "This is a sample review, with punctuations and stopwords included."
review_procesada = preprocesar_reviews(review)
print("Review original:", review)
print("Review procesada:", review_procesada)
```

```
Review original: This is a sample review, with punctuations and stopwords included.
Review procesada: sampl review punctuat stopwords includ
```

```
# Aplico el preprocesador a las reviews
datos['reviewText'] = datos['reviewText'].apply(preprocesar_reviews)
datos.head()
```

	overall	reviewerName	reviewText	summary	cardinalidad_vocabulario	sentimiento
1055801	5	Noel	larg har smallest set fit perfect stubborn eng...	EzyDog makes the best dog harness	78	Neutral
2080252	5	Laurie L. Jarvis	receiv	Five Stars	1	Neutral
1814786	5	Pamela U. Moore	product like oldest cat eat seem help right aw...	The only one that helps	41	Neutral
1167559	5	Cathy H.	need carrier stray cat ive feed take vet bough...	Perfect carrier.	91	Neutral

▼ Entrenamiento

```
# Realizo el split con variable objetivo mi columna de sentimiento
X_train_full, X_test, y_train_full, y_test = train_test_split(datos['reviewText'], datos['sentimiento'], test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.25, random_state=42)
```

```
print("El tamaño del conjunto de entrenamiento es: ", X_train.shape)
print("El tamaño del conjunto de validacion es: ", X_val.shape)
print("El tamaño del conjunto de test es: ", X_test.shape)
```

```
El tamaño del conjunto de entrenamiento es: (29992,)
El tamaño del conjunto de validacion es: (9998,)
El tamaño del conjunto de test es: (9998,)
```

```
# Creo el modelo bolsa de palabras
vectorizer = CountVectorizer(max_features=1000, binary=True) # escogí 1000 palabras
X_train_bow = vectorizer.fit_transform(X_train)
X_val_bow = vectorizer.transform(X_val) # Transformación en el conjunto de validación
X_test_bow = vectorizer.transform(X_test)
```

```
# Primero utilizaré Regresión Logística
modelo_1 = LogisticRegression(class_weight='balanced')
modelo_1.fit(X_train_bow, y_train)
```

```
# Entreno
y_pred_val = modelo_1.predict(X_val_bow)
print("Modelo 1 (Regresión Logística) - Validación:")
print(classification_report(y_val, y_pred_val))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
Modelo 1 (Regresión Logística) - Validación:
      precision    recall  f1-score   support

   Negativo      0.06      0.20      0.09         25
    Neutral      0.98      0.90      0.94       9315
    Positivo      0.37      0.77      0.50         658

   accuracy              0.89         9998
  macro avg              0.47      0.62      0.51         9998
 weighted avg              0.94      0.89      0.91         9998
```

Dado el desbalanceo considerable de mis datos creo conveniente aplicar una métrica más: el area bajo la curva

```

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

# Primero preparo mis variables objetivos a numéricos
y_train_f = y_train.replace({'Negativo': 0, 'Neutral': 1, 'Positivo': 2})
y_val_f = y_val.replace({'Negativo': 0, 'Neutral': 1, 'Positivo': 2})
y_test_f = y_test.replace({'Negativo': 0, 'Neutral': 1, 'Positivo': 2})

y_pred_proba = modelo_1.predict_proba(X_val_bow)

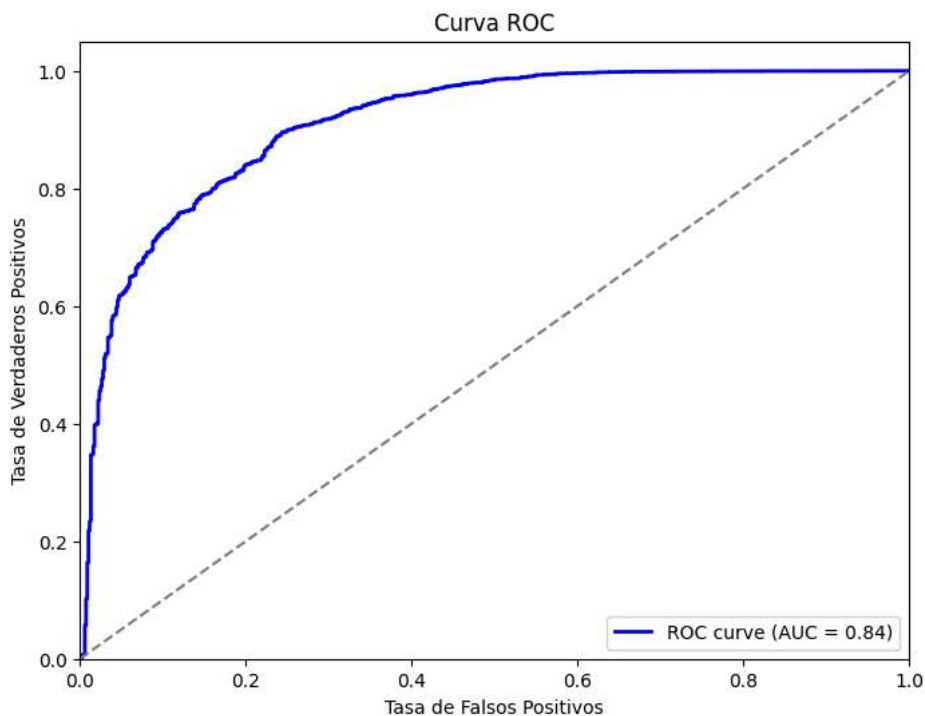
y_pred_proba_negative = y_pred_proba[:, 0]
y_pred_proba_positive = y_pred_proba[:, 2]

# Calcular el AUC-ROC para la clase Negativa, positiva y neutral
auc_roc_negative = roc_auc_score(y_val_f == 0, y_pred_proba_negative, multi_class='ovr')
auc_roc_positive = roc_auc_score(y_val_f == 2, y_pred_proba_positive, multi_class='ovr')
auc_roc = roc_auc_score(y_val_f, y_pred_proba, multi_class='ovr')

print("Área bajo la curva ROC (AUC-ROC) para la clase Negativa:", auc_roc_negative)
print("Área bajo la curva ROC (AUC-ROC) para la clase Positiva:", auc_roc_positive)
print("Área bajo la curva ROC (AUC-ROC) para la clase Neutral:", auc_roc)

Área bajo la curva ROC (AUC-ROC) para la clase Negativa: 0.6725478792740399
Área bajo la curva ROC (AUC-ROC) para la clase Positiva: 0.9401456623471294
Área bajo la curva ROC (AUC-ROC) para la clase Neutral: 0.8412362577064503

```



```

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout

# Primero preparo mis variables objetivos a numéricos
y_train_f = y_train.replace({'Negativo': 0, 'Neutral': 1, 'Positivo': 2})
y_val_f = y_val.replace({'Negativo': 0, 'Neutral': 1, 'Positivo': 2})
y_test_f = y_test.replace({'Negativo': 0, 'Neutral': 1, 'Positivo': 2})

X_train_bow_dense = X_train_bow.toarray()
X_val_bow_dense = X_val_bow.toarray()
X_test_bow_dense = X_test_bow.toarray()

modelo_2 = Sequential()
modelo_2.add(Embedding(input_dim=X_train_bow_dense.shape[1], output_dim=16, input_length=X_train_bow.shape[1]))
modelo_2.add(LSTM(units=64, dropout=0.5))
modelo_2.add(Dense(units=1, activation='softmax'))

# Compilo
modelo_2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Entreno
history = modelo_2.fit(X_train_bow_dense, y_train_f, epochs=5, batch_size=64, validation_data=(X_val_bow_dense, y_val_f))

# Evaluo
test_loss, test_accuracy = modelo_2.evaluate(X_test_bow_dense, y_test_f)
print("Accuracy en el conjunto de test:", test_accuracy)

Epoch 1/5
469/469 [=====] - 367s 770ms/step - loss: -1.3513 - accuracy: 0.9323 - val_loss: -2.2537 - val_accuracy: 0
Epoch 2/5
469/469 [=====] - 356s 760ms/step - loss: -3.1664 - accuracy: 0.9323 - val_loss: -3.9966 - val_accuracy: 0
Epoch 3/5
469/469 [=====] - 358s 763ms/step - loss: -4.9392 - accuracy: 0.9323 - val_loss: -5.7542 - val_accuracy: 0
Epoch 4/5
469/469 [=====] - 355s 758ms/step - loss: -6.7155 - accuracy: 0.9323 - val_loss: -7.4885 - val_accuracy: 0
Epoch 5/5
469/469 [=====] - 359s 765ms/step - loss: -8.4673 - accuracy: 0.9323 - val_loss: -9.2010 - val_accuracy: 0
313/313 [=====] - 39s 126ms/step - loss: -9.3028 - accuracy: 0.9328
Accuracy en el conjunto de test: 0.9327865839004517

# también aplico el area bajo la curva a este modelo

y_pred_val_probs = modelo_2.predict(X_val_bow_dense)

y_pred_val_probs_3_classes = np.concatenate([1 - y_pred_val_probs, y_pred_val_probs, np.zeros_like(y_pred_val_probs)], axis=1)

# Calcular el AUC-ROC para cada clase
auc_roc_negativo = roc_auc_score((y_val_f == 0).astype(int), y_pred_val_probs_3_classes[:, 0])
auc_roc_neutral = roc_auc_score((y_val_f == 1).astype(int), y_pred_val_probs_3_classes[:, 1])
auc_roc_positivo = roc_auc_score((y_val_f == 2).astype(int), y_pred_val_probs_3_classes[:, 2])

# Mostrar los resultados
print("Área bajo la curva ROC (AUC-ROC) para la clase Negativa:", auc_roc_negativo)
print("Área bajo la curva ROC (AUC-ROC) para la clase Neutral:", auc_roc_neutral)
print("Área bajo la curva ROC (AUC-ROC) para la clase Positiva:", auc_roc_positivo)

313/313 [=====] - 44s 139ms/step
Área bajo la curva ROC (AUC-ROC) para la clase Negativa: 0.5003008121929209
Área bajo la curva ROC (AUC-ROC) para la clase Neutral: 0.5003220611916264
Área bajo la curva ROC (AUC-ROC) para la clase Positiva: 0.5

```

Ante los resultados creo que es mejor el modelo de Regresión Logística porque considero que en mis datos hay un desequilibrio de clases significativo, como mostré anteriormente:

- En Neutral hay 46602 registros
- Positivo tiene solo 3293
- Pero Negativo es aun menor con 93 Por eso creo que me interesa saber las métricas como el f1-score o el f1-score ponderado ya que necesito un equilibrio entre precisión y recall y el modelo de regresión logística tiene un f1-score ponderado del 91%, mientras que el LSTM no proporciona estas métricas específicas. Además luego analizando las métricas del área bajo la curva que me indica que también el modelo de regresión Logística sigue siendo el mejor, esto con la consideración de que el modelo LSTM podría ser mejorable

✓ Reporte de métricas


```
# Evaluación con Regresión Logística
y_pred_test = modelo_1.predict(X_test_bow)
print("Modelo 1 (Regresión Logística) - Prueba:")
print(classification_report(y_test, y_pred_test))
```

Modelo 1 (Regresión Logística) - Prueba:				
	precision	recall	f1-score	support
Negativo	0.01	0.06	0.02	16
Neutral	0.98	0.89	0.93	9326
Positivo	0.35	0.77	0.48	656
accuracy			0.88	9998
macro avg	0.45	0.57	0.48	9998
weighted avg	0.94	0.88	0.90	9998

```

y_pred_test = modelo_1.predict_proba(X_test_bow)

y_pred_test_positivo = y_pred_test[:, 2]
y_pred_test_negativo = y_pred_test[:, 0]

# Calcular el AUC-ROC para la clase Positiva, negativa y neutral
auc_roc_negativo_test = roc_auc_score(y_test_f == 0, y_pred_test_negativo)
auc_roc_positivo_test = roc_auc_score(y_test_f == 2, y_pred_test_positivo)
auc_roc_neutral = roc_auc_score(y_test_f, y_pred_test, multi_class='ovr')

print("Área bajo la curva ROC (AUC-ROC) para la clase Negativa:", auc_roc_negative)
print("Área bajo la curva ROC (AUC-ROC) para la clase Positiva:", auc_roc_positive)
print("Área bajo la curva ROC (AUC-ROC) para la clase Neutral:", auc_roc_neutral)

Área bajo la curva ROC (AUC-ROC) para la clase Negativa: 0.5502780004007214
Área bajo la curva ROC (AUC-ROC) para la clase Positiva: 0.9382016568238901
Área bajo la curva ROC (AUC-ROC) para la clase Neutral: 0.7993714971607134

```

Aunque los modelos vistos en sí tenían buen accuracy, los datos estaban desequilibrados, por eso decidí también añadir la métrica del área bajo la curva ya que me proporciona una medida robusta del rendimiento del modelo.

En cuanto a los resultados obtenidos en el conjunto test veo que la precisión es alta para la clase Neutral (98%) y moderada para la clase Positiva (35%), pero muy baja para la clase Negativa (1%) como es de esperarse por ese desbalanceo mencionado. Sin embargo el recall es alto para la clase Positiva (77%) y moderado para la clase Neutral (89%), pero muy bajo para la clase Negativa (6%). Con el F1-score, que combina precisión y recall, ocurre lo mismo que con la precision, es bajo para la clase Negativa (2%) y moderado para la clase Positiva (48%) y Neutral (93%). En general, la precisión global del modelo (accuracy) es del 88%, lo que indica que el modelo clasifica correctamente el 88% de