Universidad Camilo José Cela

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

Procesamiento del Lenguaje Natural

Prueba de Evaluación Continua 1

Nombre y Apellidos del alumno

MILTON ASTUDILLO ASTUDILLO

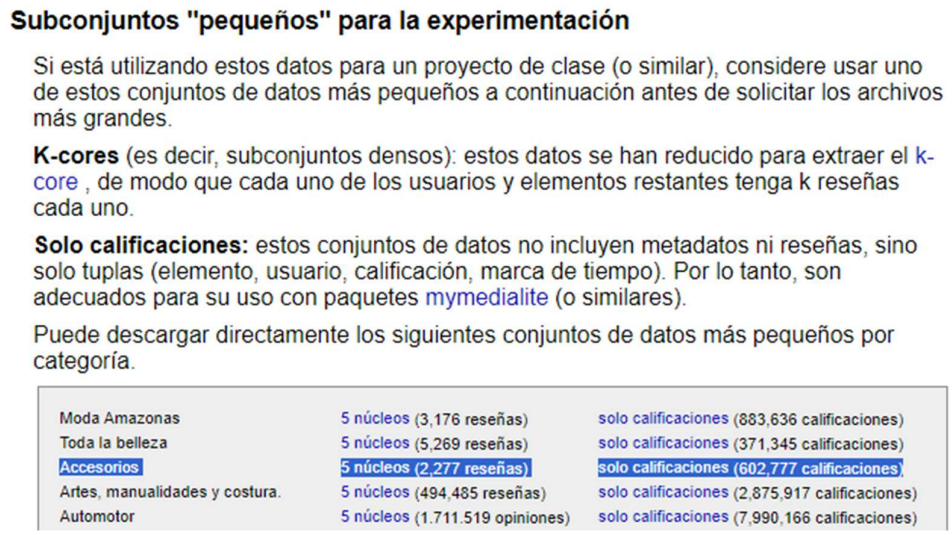
2023 - 2024

Contenido

1. Descripción del corpus y sus principales características.

El corpus con el cual decidí trabajar es parte de **Amazon Review Data (2018)** el cual se puede encontrar en este enlace https://nijianmo.github.io/amazon/index.html , después de hacer una búsqueda y elección de otros corpus posibles propuestos.

He decidido elegir un subconjunto de las reseñas de productos en la categoría sobre **accesorios** que consta de 2,277 reseñas y 602,777 calificaciones.



Subconjuntos "pequeños" para la experimentación

Si está utilizando estos datos para un proyecto de clase (o similar), considere usar uno de estos conjuntos de datos más pequeños a continuación antes de solicitar los archivos más grandes.

K-cores (es decir, subconjuntos densos): estos datos se han reducido para extraer el k-core , de modo que cada uno de los usuarios y elementos restantes tenga k reseñas cada uno.

Solo calificaciones: estos conjuntos de datos no incluyen metadatos ni reseñas, sino solo tuplas (elemento, usuario, calificación, marca de tiempo). Por lo tanto, son adecuados para su uso con paquetes mymedialite (o similares).

Puede descargar directamente los siguientes conjuntos de datos más pequeños por categoría.

Moda Amazonas	5 núcleos (3,176 reseñas)	solo calificaciones (883,636 calificaciones)
Toda la belleza	5 núcleos (5,269 reseñas)	solo calificaciones (371,345 calificaciones)
Accesorios	5 núcleos (2,277 reseñas)	solo calificaciones (602,777 calificaciones)
Artes, manualidades y costura.	5 núcleos (494,485 reseñas)	solo calificaciones (2,875,917 calificaciones)
Automotor	5 núcleos (1.711.519 opiniones)	solo calificaciones (7,990,166 calificaciones)

2. Definición detallada del objeto de clasificación

Para este corpus una tarea de clasificación adecuada y relevante sería la clasificación de la polaridad de las reseñas. En esta tarea, utilizaremos el corpus con el objetivo de clasificar las reseñas como positivas, negativas o neutrales basándote en la calificación (rating) que los usuarios dieron a los productos. Esto es útil para análisis de sentimientos, que es una aplicación común y valiosa en el análisis de reseñas de productos.

El objetivo de clasificación es determinar si de las reseñas de productos son positivas o negativas, clasificándolas en:

Positivas: Calificaciones de 4 y 5.

Negativas: Calificaciones de 1, 2 y 3.

3. Exploratoria estadística elemental del corpus

El tamaño del corpus es de 72.998 bytes

dtypes: bool(1), float64(1), int64(1), object(9) memory usage: 98.0+ KB

4. Explicación detallada de la solución implementada en cada uno de los objetivos secuenciales de la tarea.

- **Limpieza y preparación de los datos.**

```
import pandas as pd
import gzip
import json

# Ruta al archivo en tu Google Drive
file_path = '/content/drive/My Drive/Appliances_5.json.gz'

# Cargar el archivo JSON
with gzip.open(file_path, 'rt', encoding='utf-8') as f:
    data = [json.loads(line) for line in f]

# Convertir a DataFrame
df = pd.DataFrame(data)

# Mostrar las primeras filas del dataframe
print(df.head())

# Información del dataframe
print(df.info())
```

Se cargaron las bibliotecas necesarias para la manipulación de datos, manejo de archivos comprimidos y procesamiento de JSON. Se utiliza el utf-8 para su lectura. La lista de objetos JSON se convierte en un DataFrame de pandas y se muestra una vista rápida de las primeras filas del DataFrame y un resumen informativo sobre su estructura.

```
import re
import string
from sklearn.model_selection import train_test_split

# Eliminar filas con valores nulos en `reviewText` y `overall`
df_clean = df.dropna(subset=['reviewText', 'overall'])

# Convertir la columna 'overall' a tipo numérico
df_clean['overall'] = pd.to_numeric(df_clean['overall'], errors='coerce')

# Eliminar filas con valores negativos en `overall`
df_clean = df_clean[df_clean['overall'] >= 0]

# Convertir las puntuaciones `overall` a una clasificación binaria
df_clean['label'] = df_clean['overall'].apply(lambda x: 1 if x >= 4 else 0)

# Función para limpiar el texto
def clean_text(text):
    text = text.lower()  # Convertir a minúsculas
    text = re.sub(r'\d+', '', text)  # Eliminar números
    text = text.translate(str.maketrans('', '', string.punctuation))  # Eliminar puntuación
    text = text.strip()  # Eliminar espacios en blanco al inicio y al final
    return text

# Aplicar limpieza de texto
df_clean['cleaned_reviewText'] = df_clean['reviewText'].apply(clean_text)
```

-Se eliminan las filas que tengan valores nulos en las columnas reviewText (el texto de la reseña) y overall (la calificación).

-Se asegura que los valores en la columna overall sean numéricos. Si hay valores que no se pueden convertir, se reemplazan con NaN.

-Se eliminan las filas donde la calificación overall sea negativa, aunque generalmente las calificaciones suelen ser positivas, esto es una medida de limpieza adicional ya que en el proceso de Word Embeddings con el clasificador nos daba un error de numero negativo.

-Se crea una nueva columna label donde se asigna un 1 si la calificación es 4 o mayor (reseñas positivas) y un 0 si es menor (reseñas negativas).

-Definimos una función para limpiar el texto, para convertir el texto de mayúscula a minúscula, eliminar números, signos de puntuación y espacios en blanco.

-dividimos los datos en conjuntos de entrenamiento y prueba.

- **Clasificador en base a heurísticas (Regexp y lexicones)**

```
#definimos un lexicon simple
positive_words = set(['good', 'great', 'excellent', 'amazing', 'fantastic', 'love'])
negative_words = set(['bad', 'terrible', 'awful', 'worst', 'poor', 'hate'])

def heuristic_classifier(text):
    pos_count = sum([1 for word in text.split() if word in positive_words])
    neg_count = sum([1 for word in text.split() if word in negative_words])
    return 1 if pos_count > neg_count else 0

# Aplicar el clasificador heurístico al conjunto de prueba
heuristic_predictions = X_test.apply(heuristic_classifier)

# Evaluación del clasificador heurístico
from sklearn.metrics import accuracy_score

heuristic_accuracy = accuracy_score(y_test, heuristic_predictions)
heuristic_accuracy
```

```
0.618421052631579
```

-Se definen dos conjuntos (set), uno para las palabras positivas y otro para las negativas. Este léxico servirá como base para clasificar las reseñas.

-La función heuristic_classifier clasifica un texto en positivo (1) o negativo (0) de la siguiente manera:

  - Divide el texto en palabras usando split ().
  - Cuenta cuántas palabras pertenecen al conjunto de palabras positivas (pos_count).
  - Cuenta cuántas palabras pertenecen al conjunto de palabras negativas (neg_count).
  - Devuelve 1 si el número de palabras positivas es mayor que el de palabras negativas; de lo contrario, devuelve 0.

-Se aplica la función heuristic_classifier a cada texto del conjunto de prueba (X_test) usando apply(). El resultado es una serie (Series) de predicciones heurísticas.

- **Representación del texto mediante BoW y clasificador Naive Bayes**

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

# Representación BoW
vectorizer = CountVectorizer()
X_train_bow = vectorizer.fit_transform(X_train)
X_test_bow = vectorizer.transform(X_test)

# Clasificador Naive Bayes
nb_classifier = MultinomialNB()
nb_classifier.fit(X_train_bow, y_train)
nb_predictions = nb_classifier.predict(X_test_bow)

# Evaluación del clasificador Naive Bayes
nb_accuracy = accuracy_score(y_test, nb_predictions)

# Clasificador de Regresión Logística
lr_classifier = LogisticRegression(max_iter=1000)
lr_classifier.fit(X_train_bow, y_train)
lr_predictions = lr_classifier.predict(X_test_bow)

# Evaluación del clasificador de Regresión Logística
lr_accuracy = accuracy_score(y_test, lr_predictions)

nb_accuracy, lr_accuracy
```

- Convertir el texto a una representación de bolsa de palabras

CountVectorizer transforma el texto en una matriz de características, donde cada fila representa un documento y cada columna representa una palabra del vocabulario.

fit_transform se aplica al conjunto de entrenamiento (X_train) para aprender el vocabulario y transformar el texto a una matriz de características.

transform se aplica al conjunto de prueba (X_test) para transformar el texto en base al vocabulario aprendido.

- Entrenar y evaluar el clasificador Naive Bayes

MultinomialNB se utiliza para clasificación de texto, siendo adecuado para datos discretos como la representación BoW. **fit** entrena el modelo con los datos de entrenamiento. Y **predict** genera predicciones para los datos de prueba.

- Entrenar y evaluar el clasificador de Regresión Logística

LogisticRegression se utiliza para clasificación binaria. Se especifica max_iter=1000 para asegurar la convergencia del algoritmo. **fit** entrena el modelo con los datos de entrenamiento y **predict** genera predicciones para los datos de prueba.

```python
import numpy as np
from gensim.models import Word2Vec
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Crear embeddings Word2Vec sobre el propio corpus
sentences = [text.split() for text in X_train]
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=2, workers=4)
w2v_model.train(sentences, total_examples=len(sentences), epochs=10)

# Función para obtener los embeddings de Word2Vec
def get_w2v_embeddings(text):
    words = text.split()
    embedding = np.mean([w2v_model.wv[word] for word in words if word in w2v_model.wv] or [np.zeros(100)], axis=0)
    return embedding

# Convertir los textos a sus embeddings
X_train_w2v = np.array([get_w2v_embeddings(text) for text in X_train])
X_test_w2v = np.array([get_w2v_embeddings(text) for text in X_test])

# Clasificador de Regresión Logística
lr_w2v_classifier = LogisticRegression(max_iter=1000)
lr_w2v_classifier.fit(X_train_w2v, y_train)
lr_w2v_predictions = lr_w2v_classifier.predict(X_test_w2v)

# Evaluación del clasificador de Regresión Logística con Word Embeddings
lr_w2v_accuracy = accuracy_score(y_test, lr_w2v_predictions)
```

-Se convierte cada texto del conjunto de entrenamiento (X_train) en una lista de palabras (oraciones).
-Se crea un modelo Word2Vec con las siguientes especificaciones:

vector_size=100: Dimensión de los embeddings.
window=5: Tamaño de la ventana de contexto.
min_count=2: Palabras con frecuencia menor a 2 se ignoran.
workers=4: Número de hilos para entrenamiento.

- El modelo se entrena con las oraciones preparadas (sentences) durante 10 épocas.

- La función get_w2v_embeddings convierte un texto en su embedding correspondiente,

Divide el texto en palabras. Obtiene los embeddings de cada palabra (si existe en el vocabulario de Word2Vec). Calcula la media de los embeddings de las palabras para obtener un solo vector representativo del texto. Si ninguna palabra está en el vocabulario, se devuelve un vector de ceros.

- Se aplica la función get_w2v_embeddings a cada texto del conjunto de entrenamiento (X_train) y del conjunto de prueba (X_test) para obtener sus representaciones en forma de embeddings.

- Entrenar el clasificador de Regresión Logística

  Se crea un clasificador de Regresión Logística (LogisticRegression) con un máximo de 1000 iteraciones.
  Se entrena (fit) el clasificador con los embeddings de entrenamiento (X_train_w2v) y sus etiquetas (y_train).
  Se generan predicciones (predict) para los embeddings del conjunto de prueba (X_test_w2v).

- **Clasificador con WE y una arquitectura de DL.**

```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, SpatialDropout1D
from sklearn.metrics import accuracy_score

# Tokenización y padding
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Definir la longitud máxima de las secuencias
max_len = max(len(seq) for seq in X_train_seq)

# Aplicar padding
X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

# Crear una matriz de embeddings
word_index = tokenizer.word_index
embedding_matrix = np.zeros((len(word_index) + 1, 100))
for word, i in word_index.items():
    if word in w2v_model.wv:
        embedding_matrix[i] = w2v_model.wv[word]
```

- Tokenización y padding

  Tokenizer convierte el texto en secuencias de enteros, donde cada entero representa una palabra.
  fit_on_texts ajusta el tokenizador a los textos de entrenamiento (X_train), creando un índice de palabras.
  texts_to_sequences convierte los textos en secuencias de enteros basadas en el índice de palabras.

- Definir la longitud máxima de las secuencias

  Se calcula la longitud máxima de las secuencias en el conjunto de entrenamiento (X_train_seq). Esta longitud se utilizará para el padding.

- Aplicar padding

  pad_sequences se utiliza para asegurar que todas las secuencias tengan la misma longitud (max_len) rellenando con ceros si es necesario.

- Crear una matriz de embeddings

  word_index contiene el índice de cada palabra ajustado en el tokenizador.

  embedding_matrix es una matriz donde cada fila representa el vector de embedding para una palabra del vocabulario. Si una palabra no está en el modelo Word2Vec, su vector será un vector de ceros.

- Definir y compilar el modelo

  Embedding inicializa la capa de embeddings con embedding_matrix y asegura que los embeddings no sean entrenables (trainable=False).

SpatialDropout1D aplica dropout a las entradas de la capa LSTM para evitar el sobreajuste.

LSTM es la capa de LSTM con dropout en las conexiones recurrentes.

Dense con activación sigmoid para la clasificación binaria.

El modelo se compila con la función de pérdida **binary_crossentropy** y el optimizador **adam**.

Luego se entrena y evalúa el modelo.

Construcción del Modelo LSTM

```
# Construir el modelo LSTM
model = Sequential()
model.add(Embedding(input_dim=len(word_index) + 1, output_dim=100, weights=[embedding_matrix], input_length=max_len, trainable=False))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Entrenar el modelo
history = model.fit(X_train_pad, y_train, epochs=5, batch_size=64, validation_data=(X_test_pad, y_test), verbose=2)

# Evaluación del modelo
dl_accuracy = model.evaluate(X_test_pad, y_test, verbose=0)[1]
print(f"Accuracy of LSTM with Word Embeddings: {dl_accuracy}")
```

Definición del modelo

Sequential: Se utiliza para definir una pila lineal de capas.

Embedding: Inicializa la capa de embeddings con embedding_matrix.

input_dim es el tamaño del vocabulario (número de palabras + 1 para el índice 0).

output_dim es la dimensión de los embeddings (100 en este caso).

weights se inicializa con los embeddings preentrenados.

input_length define la longitud máxima de las secuencias de entrada.

trainable=False indica que los embeddings no se actualizarán durante el entrenamiento.

SpatialDropout1D: Aplicar dropout a las entradas de la capa LSTM para reducir el sobreajuste.

LSTM: Añade una capa LSTM con 100 unidades.

dropout=0.2 y recurrent_dropout=0.2 aplican dropout regular y recurrente para evitar el sobreajuste.

Dense: Añade una capa densa con una unidad de salida y activación sigmoid para la clasificación binaria.

compile: Compila el modelo con la función de pérdida binary_crossentropy, el optimizador adam y la métrica de accuracy

5. presentación y descripción de las librerías y algoritmos

5.1. Limpieza y Preparación de los Datos

Librerías Utilizadas:

- re: Utilizada para operaciones de expresiones regulares, facilitando la limpieza del texto al eliminar números y caracteres especiales.

- string: Utilizada para la manipulación de cadenas de caracteres, especialmente para eliminar puntuaciones.

- sklearn.model_selection (train_test_split) : Utilizada para dividir el conjunto de datos en conjuntos de entrenamiento y prueba, asegurando que el modelo se entrene y se pruebe en datos diferentes.

Descripción:

Se realiza la limpieza de datos eliminando valores nulos y negativos, se convierte la columna de calificaciones a una clasificación binaria (positivo o negativo), y se limpia el texto de las reseñas eliminando puntuaciones y números. Finalmente, los datos se dividen en conjuntos de entrenamiento y prueba.

5.2. Representación del Texto mediante Word Embeddings

Librerías Utilizadas:

- gensim.models.Word2Vec : Utilizada para crear modelos de Word2Vec, generando representaciones vectoriales de palabras (embeddings) entrenadas sobre el propio corpus.

Descripción:

Se crean embeddings de Word2Vec a partir de las reseñas textuales, representando cada palabra como un vector en un espacio de alta dimensión. Estos embeddings capturan relaciones semánticas entre las palabras, mejorando la capacidad del modelo para comprender y procesar el texto.

5.3. Clasificación con Regresión Logística y Word Embeddings

Librerías Utilizadas:

- numpy : Utilizada para operaciones numéricas, facilitando la manipulación de matrices y arrays.

- sklearn.linear_model (LogisticRegression) : Utilizada para construir y entrenar un clasificador de Regresión Logística, adecuado para la clasificación binaria.

Descripción:

Los textos representados mediante embeddings se utilizan para entrenar un modelo de Regresión Logística. Este modelo es adecuado para datos con valores negativos y aprovecha los embeddings para realizar predicciones sobre la polaridad de las reseñas.

5.4. Clasificación con Word Embeddings y Arquitectura de Deep Learning (LSTM)

Librerías Utilizadas:

- tensorflow.keras.preprocessing.text (Tokenizer, pad_sequences) : Utilizadas para la tokenización del texto y el padding de secuencias, preparando los datos textuales para ser introducidos en la red neuronal.

- tensorflow.keras.models (Sequential) : Utilizada para construir modelos secuenciales de redes neuronales.

- tensorflow.keras.layers (Embedding, LSTM, Dense, SpatialDropout1D) : Utilizadas para crear y añadir capas al modelo de red neuronal, incluyendo una capa de embeddings, una capa LSTM y una capa densa para la clasificación.

- sklearn.metrics (accuracy_score) : Utilizada para evaluar el rendimiento del modelo mediante la métrica de precisión.

Descripción:

Se construye y entrena una red neuronal recurrente con LSTM utilizando embeddings de Word2Vec. Este modelo es capaz de capturar dependencias a largo plazo en el texto, mejorando la precisión en la clasificación de las reseñas.

6. Justificación en la Selección de las Librerías y Algoritmos

6.1. Limpieza y Preparación de los Datos

Librerías Utilizadas:

- re: Utilizada para operaciones de expresiones regulares que facilitan la limpieza de texto, como la eliminación de números y caracteres especiales. Es fundamental para la preparación de texto sin ruido.

- string: Utilizada para la eliminación de puntuaciones.

6.2. Representación del Texto mediante Word Embeddings

- gensim.models.Word2Vec : Seleccionada por su eficacia en la creación de representaciones vectoriales de palabras (embeddings). Word2Vec es conocido por capturar relaciones semánticas entre palabras, lo que mejora la comprensión del contexto en los modelos de lenguaje.

Word2Vec proporciona una representación densa de palabras que captura similitudes semánticas, lo que es esencial para tareas de procesamiento de lenguaje natural. Entrenar los embeddings en el propio corpus asegura que las representaciones sean específicas y relevantes para el dominio de los datos.

6.3. Clasificación con Regresión Logística y Word Embeddings

- sklearn.linear_model (LogisticRegression) : Seleccionada por ser un algoritmo simple y eficiente para problemas de clasificación binaria. La regresión logística es robusta y fácil de interpretar, y funciona bien con los embeddings de Word2Vec que pueden contener valores negativos.

La regresión logística es una técnica clásica que proporciona una buena línea base para la clasificación binaria. Su uso combinado con embeddings de Word2Vec permite aprovechar las representaciones densas de las palabras para mejorar el rendimiento del modelo.

6.4. Clasificación con Word Embeddings y Arquitectura de Deep Learning (LSTM)

- tensorflow.keras.preprocessing.text (Tokenizer, pad_sequences) : Seleccionadas para convertir texto en secuencias de enteros y asegurar que todas las secuencias tengan la misma longitud, lo cual es crucial para la entrada en redes neuronales.

- tensorflow.keras.models (Sequential) : Elegida por su simplicidad en la construcción de modelos secuenciales de redes neuronales.

- tensorflow.keras.layers (Embedding, LSTM, Dense, SpatialDropout1D) : Utilizadas para construir un modelo LSTM. La capa de embeddings inicializa con los embeddings de Word2Vec, LSTM captura dependencias temporales, Dense realiza la clasificación, y SpatialDropout1D previene el sobreajuste.

- sklearn.metrics (accuracy_score) : Utilizada para evaluar el rendimiento del modelo mediante la métrica de precisión, proporcionando una medida clara de la exactitud del modelo.

Las arquitecturas de redes neuronales, especialmente LSTM, son adecuadas para modelar secuencias de texto debido a su capacidad para capturar dependencias a largo plazo. Utilizar embeddings preentrenados mejora el entendimiento del modelo sobre las relaciones semánticas de las palabras. Keras facilita la implementación y experimentación con diferentes arquitecturas de redes neuronales.

7. Detalle de los Resultados Obtenidos

Enfoque	Algoritmo	Representación del Texto	Exactitud (Accuracy)
BoW + Clasificador Naive Baye	Clasificador Naive Baye	Bag of Words	0.984
BoW + Regresión Logística	Regresión Logística	Bag of Words	0.995
WE + Regresión Logística	Regresión Logística	Word Embeddings	0.995
WE + Deep Learning (LSTM)	LSTM	Word Embeddings	0.993

8. Descripción de los Problemas Encontrados y las Soluciones Adoptadas

- Problema: Datos con Valores Negativos

Descripción:
Durante la limpieza de datos, se detectaron valores negativos en la columna overall, que representa la calificación de las reseñas. Estos valores no son válidos para la clasificación de reseñas.
Solución:
Se decidió eliminar las filas con valores negativos en la columna overall. Esta limpieza asegura que solo se consideren calificaciones válidas para el análisis y la clasificación.

- Problema: Tokenización y Secuencias de Longitud Variable

Descripción:
Para entrenar redes neuronales, las secuencias de texto deben tener la misma longitud. Las reseñas tenían longitudes variables, lo que complicaba su uso directo en modelos de Deep Learning.
Solución:
Se utilizó la tokenización para convertir el texto en secuencias de enteros y se aplicó padding para asegurar que todas las secuencias tengan la misma longitud. Esto facilita la entrada de los datos en los modelos de Deep Learning y asegura que las representaciones sean consistentes.

- Problema: Valores Negativos en Embeddings

Descripción:
El clasificador Naive Bayes no acepta valores negativos, pero los embeddings de Word2Vec pueden contener valores negativos. Esto impide el uso directo de Naive Bayes con embeddings.

Solución:
Se descartó el uso de Naive Bayes con Word Embeddings y se optó por Regresión Logística, que puede manejar valores negativos en los datos de entrada. La Regresión Logística es adecuada para trabajar con las representaciones densas de Word2Vec.

- Problema: Overfitting en el Modelo LSTM

Descripción:
Los modelos de redes neuronales complejas como LSTM tienen una alta capacidad de modelado, lo que puede llevar a un sobreajuste si no se maneja adecuadamente. El sobreajuste ocurre

cuando el modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a datos no vistos.

Solución:
Se implementaron técnicas para prevenir el sobreajuste, como el uso de dropout y técnicas de validación cruzada para monitorear el rendimiento del modelo en datos de validación. Estas técnicas ayudan a asegurar que el modelo mantenga un buen rendimiento general y no se ajuste demasiado a los datos de entrenamiento.