Proyecto Aprendizaje Automático

Diego Perea
Facultad de Ingeniería,
Universidad Autónoma de Occidente
Cali, Colombia
diego.perea@uao.edu.co

Samir Hassan
Facultad de Ingeniería,
Universidad Autónoma de Occidente
Cali, Colombia

Gabriel Jeannot Viaña
Facultad de Ingeniería,
Universidad Autónoma de Occidente
Cali, Colombia
gabriel.jeannot@uao.edu.co

Carlos Osorio
Facultad de Ingeniería,
Universidad Autónoma de Occidente
Cali, Colombia

Luis Pareja
Facultad de Ingeniería,
Universidad Autónoma de Occidente
Cali, Colombia

Abstract - A review sentiment classification model is implemented using different preprocessing and training methods. The IMDB dataset, which consists of movie reviews labeled as positive or negative, is used. First, the dataset is loaded and preprocessed. Then, the model is built, compiled and trained using the training set. Finally, the model's performance on the training and test sets is evaluated and used to predict the sentiment of a user-entered review.

Keywords –

Resumen - Se implementa un modelo de clasificación de sentimiento de reseñas utilizando diferentes 4 métodos de preprocesamiento y entrenamiento. Se utiliza el conjunto de datos IMDB dataset, que consiste en reseñas de películas etiquetadas como positivas o negativas. Primero, se carga y se preprocesa el conjunto de datos. Luego, se construye el modelo, se compila y se entrena utilizando el conjunto de entrenamiento. Finalmente, se evalúa el rendimiento del modelo en los conjuntos de entrenamiento y prueba y se utiliza para predecir el sentimiento de una reseña ingresada por el usuario.

Palabras Claves - Aprendizaje automatico – método-preprocesamiento-entrenamiento

INTRODUCCIÓN

El análisis de sentimiento es una tarea importante en la minería de texto y el procesamiento del lenguaje natural. Se implementa un modelo de clasificación de sentimiento utilizando diferentes métodos de preprocesamiento y de entrenamiento. El conjunto de datos utilizado es el IMDB dataset, que contiene reseñas de películas etiquetadas como positivas o negativas.

Se clasifica reseñas de películas etiquetadas como positivas o negativas en este caso 4 métodos diferentes de preprocesamiento de los datos y diferentes modelos para su entrenamiento, con esto se graficará el loss (pérdida) y accuracy (precisión) dado a eso se mostrará la matriz de confusión que nos dará la evaluación del modelo de clasificación de reviews positivas y negativas. Para la comprobación inmediata del modelo de insertará una review nueva creada por nosotros y se utilizará el modelo creado para que realice su clasificación de una review en dos grupos, positiva o negativa.

I. MARCO TEORICO

La inteligencia artificial (IA) es un campo que se enfoca en el desarrollo de sistemas que pueden aprender y realizar tareas que normalmente requieren inteligencia humana. El entrenamiento de modelos de aprendizaje automático es fundamental en la IA, ya que permite que los sistemas puedan aprender de los datos y mejorar su capacidad para realizar tareas específicas.
Es importante tener un modelo de IA porque permite automatizar tareas que antes requerían la intervención humana, como la clasificación de imágenes, la detección de fraudes, la predicción de ventas, entre otras. Los modelos de IA pueden procesar grandes cantidades de datos de manera eficiente y proporcionar resultados precisos y consistentes.
El entrenamiento de modelos de IA es importante porque permite que el modelo aprenda de los datos de entrenamiento y se ajuste a los patrones presentes en los datos. Durante el entrenamiento, el modelo ajusta sus parámetros para minimizar la diferencia entre las predicciones y las etiquetas de los datos de entrenamiento. Una vez entrenado, el modelo puede ser utilizado para hacer predicciones en nuevos datos.

El aprendizaje automático es una rama de la inteligencia artificial que se enfoca en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender y mejorar su rendimiento a partir de datos y experiencias pasadas, sin necesidad de ser programadas de manera explícita.

En el contexto del aprendizaje automático, los algoritmos son diseñados para analizar grandes conjuntos de datos y encontrar patrones, relaciones y regularidades ocultas que pueden utilizarse para realizar tareas específicas o tomar decisiones. Estos algoritmos son capaces de aprender de manera autónoma, ajustando sus parámetros y mejorando su rendimiento a medida que se les proporciona más información. Este se basa en la idea de que las máquinas pueden aprender de los datos, identificar tendencias y realizar predicciones o tomar decisiones basadas en estos patrones identificados. Este enfoque permite abordar problemas complejos y realizar tareas que de otra manera serían difíciles o impracticables de manera manual.
El proceso de aprendizaje automático generalmente implica varias etapas, que incluyen la recopilación y preparación de datos, la selección y diseño de algoritmos apropiados, el entrenamiento y ajuste de los modelos, y la evaluación y validación de su desempeño. A medida que el modelo se expone a más datos, se espera que su capacidad para generalizar y realizar predicciones precisas mejore.

El aprendizaje automático se aplica en una amplia variedad de campos y disciplinas, incluyendo reconocimiento de voz, visión por computadora, procesamiento del lenguaje natural, medicina, finanzas, comercio electrónico, entre otros. Sus aplicaciones abarcan desde la detección de fraudes, recomendación de productos, diagnóstico médico, hasta la conducción autónoma de vehículos.

El preprocesamiento de los datos es una parte esencial de cualquier modelo de clasificación de texto; El preprocesamiento de los datos es una etapa crucial en el procesamiento del lenguaje natural y en particular en el análisis de sentimiento. La razón por la que el preprocesamiento es importante es porque los datos de texto tienen una estructura compleja y ruidosa, lo que dificulta su procesamiento por parte de los modelos de aprendizaje automático. El preprocesamiento ayuda a simplificar y estandarizar los datos de entrada, lo que permite que los modelos de aprendizaje automático puedan trabajar de manera más efectiva.
En el caso específico de la clasificación de sentimiento, el preprocesamiento se utiliza para transformar las reseñas de texto en una representación numérica que pueda ser utilizada por el modelo de aprendizaje automático. Esto implica transformar el texto en una secuencia de palabras (tokens), que luego se convierten en vectores numéricos. Además, se puede realizar la eliminación de palabras comunes, la corrección ortográfica, la eliminación de signos de puntuación, entre otras técnicas de limpieza de texto.

La clasificación de sentimientos mediante modelos de machine learning es una técnica utilizada para determinar la polaridad emocional de un texto o una frase, es decir, si expresa sentimientos positivos, negativos o neutros. Esta tarea es importante en diversas aplicaciones, como el análisis de opiniones en redes sociales, la evaluación de comentarios de productos, la detección de opiniones políticas, entre otros.

A continuación, se describen brevemente los diferentes métodos de aprendizaje automático utilizados dentro del proyecto en cuestión.

Long Short-Term Memory (LSTM) es un tipo de arquitectura de red neuronal recurrente (RNN) utilizada en el campo del Aprendizaje Automático (Machine Learning) y el Procesamiento del Lenguaje Natural (NLP). A diferencia de las redes neuronales convencionales, que tienen dificultades para procesar secuencias de datos, las LSTM están diseñadas específicamente para capturar y recordar patrones a largo plazo en secuencias. Se destacan por su capacidad para mantener y recordar información a largo plazo, evitando los problemas de desvanecimiento y explosión de gradientes que pueden surgir en las RNN tradicionales. Logran esto utilizando una estructura interna compuesta por celdas de memoria y compuertas.

La regresión logística es un algoritmo de aprendizaje supervisado que se utiliza para la clasificación de datos en problemas binarios o multiclase. Mediante el uso de la función logística, modela la relación entre las variables independientes y la probabilidad de pertenencia a una clase específica. Su aplicación versátil y su capacidad para proporcionar probabilidades de clasificación la convierten en una herramienta valiosa en el campo del machine learning.

SVM (Support Vector Machine) con matrices TF-IDF (Term Frequency-Inverse Document Frequency) es una técnica utilizada en el campo del aprendizaje automático para la clasificación de textos. El SVM es un algoritmo de aprendizaje supervisado que se utiliza para realizar tareas de clasificación y regresión. Su objetivo principal es encontrar un hiperplano en un espacio dimensional superior que permita separar de manera

óptima las diferentes clases de datos. En el contexto de la clasificación de textos, el SVM se utiliza para asignar etiquetas o categorías a documentos basados en características extraídas de ellos. La matriz TF-IDF es una representación numérica de un conjunto de documentos que captura la importancia de cada término en cada documento. TF (Term Frequency) se refiere a la frecuencia del término en un documento específico, mientras que IDF (Inverse Document Frequency) mide la importancia de un término en el conjunto de documentos. La matriz TF-IDF asigna valores numéricos a cada término en cada documento, donde los valores más altos indican una mayor relevancia.

Al combinar SVM con matrices TF-IDF, se crea un modelo de clasificación de textos que utiliza la matriz TF-IDF como características de entrada para entrenar el SVM. El SVM busca encontrar un hiperplano que separe de manera óptima los documentos en diferentes categorías, basándose en las características proporcionadas por la matriz TF-IDF. El modelo resultante puede utilizarse para clasificar nuevos documentos en las categorías definidas durante el entrenamiento.

Padding y embedding son técnicas utilizadas en el campo del machine learning, especialmente en el procesamiento del lenguaje natural (NLP), para manejar y representar texto de manera efectiva. Estas técnicas desempeñan un papel crucial en la preparación y representación de datos de texto para su posterior procesamiento y análisis.

El padding se refiere al proceso de agregar valores o elementos especiales a las secuencias de texto para que todas tengan la misma longitud. En NLP, es común trabajar con secuencias de palabras de diferentes longitudes, pero muchos modelos de machine learning requieren que los datos de entrada tengan dimensiones uniformes. Por lo tanto, se agrega padding (relleno) al final de las secuencias más cortas para igualar la longitud de las más largas. Por lo general, se utiliza un token especial, como "<PAD>", para indicar el padding. Esta técnica garantiza que todas las secuencias tengan la misma longitud, lo que facilita el procesamiento y la aplicación de algoritmos de machine learning. Por otro lado, embedding se refiere a la representación vectorial de palabras o secuencias de texto. A diferencia de los modelos tradicionales que tratan el texto como una secuencia de caracteres o palabras, los embeddings permiten asignar vectores numéricos a palabras o frases completas. Estos vectores representan de manera semántica las relaciones y similitudes entre las palabras en un espacio vectorial de alta dimensión.

II. METODOLOGÍA.

Se carga el dataset de IMDB que contiene reseñas de películas etiquetadas como positivas o negativas. A partir de esto, se realiza el primer método

Primer método: Long short-term memory (LSTM)

Se importan las bibliotecas necesarias para cargar y procesar los datos, y construir el modelo.

```
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
import numpy as np
```

Este código carga los datos del conjunto de datos IMDB y limita el vocabulario a las 10,000 palabras más frecuentes. A continuación, realiza el padding de las secuencias para que todas tengan la misma longitud. Después, se crea un modelo con una capa de embedding, una capa LSTM y una capa densa con una función de activación sigmoide. El modelo se compila con el optimizador Adam y se utiliza la métrica de pérdida binary_crossentropy y accuracy.

```
# Cargar los datos de IMDB y limitar el vocabulario a las 10,000 palabras más frecuentes
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

# Realizar padding de las secuencias para que todas tengan la misma longitud
maxlen = 500
x_train_pad = pad_sequences(x_train, maxlen=maxlen, padding='post', truncating='post')
x_test_pad = pad_sequences(x_test, maxlen=maxlen, padding='post', truncating='post')

# Crear el modelo
model = Sequential([
    Embedding(input_dim=10000, output_dim=32, input_length=maxlen),
    LSTM(32),
    Dense(1, activation='sigmoid')
])

# Compilar el modelo
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

El modelo se entrena durante 10 épocas con un tamaño de lote de 128 y se utiliza EarlyStopping como callback. Finalmente, el modelo se evalúa en los datos de prueba y se muestra la pérdida y la precisión del modelo. Además, se guarda el modelo entrenado en un archivo llamado "modelo.h5".

```
# Compilar el modelo
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Entrenar el modelo
history = model.fit(x_train_pad, y_train, epochs=10, batch_size=128, validation_data=(x_test_pad, y_test), callbacks=[EarlyStopping(patience=2)])

# Evaluar el modelo en los datos de prueba
loss, accuracy = model.evaluate(x_test_pad, y_test)

print(f'Model 1 - Test loss: {loss:.3f}, Test accuracy: {accuracy:.3f}')
# Guardar el modelo entrenado
model.save('modelo.h5')
```

Este modelo es un modelo de clasificación binaria que se utiliza para predecir si una crítica de cine es positiva o negativa. La entrada del modelo son las secuencias de palabras que se han procesado previamente y se han convertido en secuencias de números enteros. El modelo utiliza una capa de embedding para convertir estos números enteros en vectores de baja dimensión que representan cada palabra. A continuación, se utiliza una capa LSTM para procesar estas secuencias de vectores y capturar la información temporal. Finalmente, una capa densa se utiliza para producir la salida del modelo, que es una probabilidad de que la crítica de cine sea positiva o negativa.

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 1s 0us/step
Epoch 1/10
196/196 [==============================] - 118s 583ms/step - loss: 0.6930 - accuracy: 0.5016 - val_loss: 0.6937 - val_accuracy: 0.5053
Epoch 2/10
196/196 [==============================] - 115s 590ms/step - loss: 0.6905 - accuracy: 0.5170 - val_loss: 0.6919 - val_accuracy: 0.5086
Epoch 3/10
196/196 [==============================] - 114s 585ms/step - loss: 0.6834 - accuracy: 0.5232 - val_loss: 0.6925 - val_accuracy: 0.5078
Epoch 4/10
196/196 [==============================] - 114s 583ms/step - loss: 0.6675 - accuracy: 0.5355 - val_loss: 0.6960 - val_accuracy: 0.5154
782/782 [==============================] - 41s 53ms/step - loss: 0.6960 - accuracy: 0.5154
Model 1 - Test loss: 0.696, Test accuracy: 0.515
```

Este código utiliza la librería matplotlib para generar dos gráficas que muestran la evolución del loss y la accuracy durante el entrenamiento de un modelo de machine learning. Además, utiliza la función confusion_matrix de la librería sklearn.metrics para generar una matriz de confusión que muestra la cantidad de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos obtenidos por el modelo en los datos de prueba.

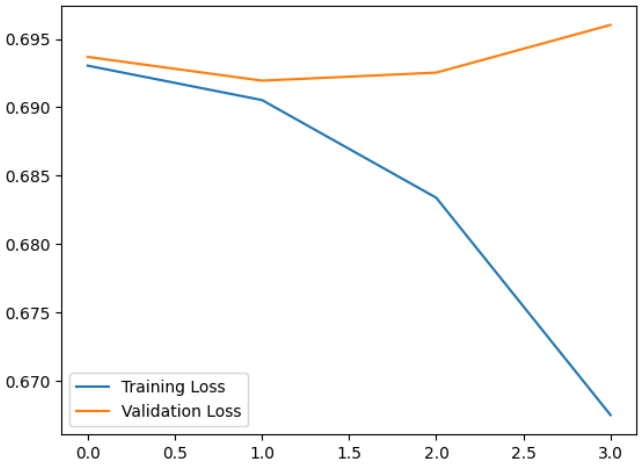
```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
```

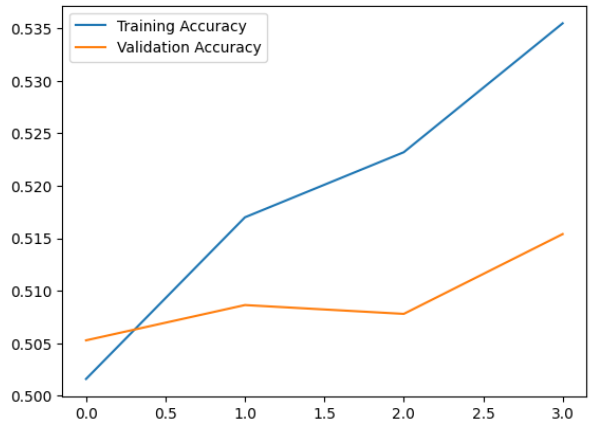
En cuanto al código específico, las primeras líneas obtienen las predicciones del modelo en los datos de prueba y convierten las probabilidades a etiquetas binarias utilizando la función np.round. Luego, se calcula la matriz de confusión utilizando la función confusion_matrix de sklearn.metrics.

```
# Obtener las predicciones del modelo en los datos de prueba
y_pred = model.predict(x_test_pad)

# Convertir las probabilidades a etiquetas binarias
y_pred = np.round(y_pred)

# Obtener la matriz de confusión
cm = confusion_matrix(y_test, y_pred)
```

Las siguientes dos líneas utilizan la librería matplotlib para generar dos gráficas que muestran la evolución del loss y la accuracy durante el entrenamiento. En la primera gráfica se muestra la evolución del loss tanto para los datos de entrenamiento como para los datos de validación, mientras que en la segunda se muestra la evolución de la accuracy.
Finalmente, se imprime la matriz de confusión utilizando la función print. Esta matriz muestra la cantidad de verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos obtenidos por el modelo en los datos de prueba.

```
# Graficar la evolución del loss y accuracy durante el entrenamiento
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()

# Imprimir la matriz de confusión
print('Confusion Matrix:')
print(cm)
```






```
Confusion Matrix:
[[12039   461]
 [11654   846]]
```

Luego se solicita al usuario que ingrese una reseña para clasificarla como positiva o negativa utilizando un modelo de machine learning entrenado con el dataset IMDB. El proceso de clasificación se realiza en unos pocos pasos:

- Se solicita al usuario que ingrese una reseña a través de la función input.
- Se tokeniza la reseña del usuario utilizando el diccionario de palabras del dataset IMDB, el cual se obtiene con la función get_word_index. Si una palabra de la reseña no se encuentra en el diccionario, se asigna un valor de 0.
- Se acorta o se extiende la reseña del usuario hasta un máximo de 256 palabras utilizando la función pad_sequences, la cual completa las secuencias con ceros al principio o al final según sea necesario.
- Se utiliza el modelo entrenado para predecir el sentimiento de la reseña del usuario utilizando la función predict. La salida de la función es un número entre 0 y 1 que representa la probabilidad de que la reseña sea positiva.

```
# Prompt the user to enter a review for classification
user_review = input("Enter a review for classification: ")

# Tokenize the user's review using the IMDB dataset's word index
word_index = imdb.get_word_index()
user_tokens = [word_index[word] if word in word_index else 0 for word in user_review.split()]

# Pad the user's tokens to a maximum length of 256 words
user_padded = pad_sequences([user_tokens], maxlen=maxlen)

# Use the trained model to predict the sentiment of the user's review
prediction = model.predict(user_padded)[0][0]

# Print the predicted sentiment
if prediction >= 0.5:
    print(f"Prediction: {prediction}")
    print("Positive review!")
else:
    print(f"Prediction: {prediction}")
    print("Negative review.")
```

```
Enter a review for classification: nice movie
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 1s 0us/step
1/1 [==============================] - 0s 52ms/step
Prediction: 0.5071957111358643
Positive review!
```

Segundo método: Regresión logística

Mediante este método se buscó realizar el entrenamiento de un modelo que permita la clasificación de reseñas de películas mediante un algoritmo de machine learning con regresión logística. Para contextualizar mejor debemos saber el concepto mínimo de regresión logística y la diferencia con su homólogo más relevante y conocido como lo es, la regresión lineal. La regresión logística es un método estadístico utilizado para modelar la relación entre variables predictoras y una variable de resultado binaria. A diferencia de la regresión lineal, que se utiliza para variables continuas, la regresión logística se aplica cuando la variable dependiente tiene dos categorías distintas, como "sí/no" o "bueno/malo".

En el caso del contexto inmediato, buscando clasificar un data base de reseñas de películas entre buenas y malas, la regresión logística es completamente viable siempre que haya un conjunto de datos con reseñas de películas, donde cada reseña esté etiquetada como "buena" o "mala". Luego de ello, se pueden utilizar características de las reseñas (como el contenido textual, la calificación, el género de la película, etc.) como variables predictoras. En esta ocasión se limitará netamente a trabajar con el contenido textual de la reseña.

Aun así, es importante tener en cuenta, como en todo modelo de aprendizaje supervisado, que la eficacia de la regresión logística para clasificar reseñas de películas dependerá de la calidad y representatividad del conjunto de datos, así como de las características seleccionadas o variables predictoras. Por tanto, luego del entrenamiento y testeo, es recomendable realizar una evaluación rigurosa del modelo utilizando métricas de desempeño adecuadas, como la precisión, o la exhaustividad.

Siguiente a ello, se presenta una explicación en la implementación y desarrollo del algoritmo que potencia este modelo para clasificación de textos.

En estas líneas, se importan las bibliotecas necesarias: numpy para operaciones numéricas, imdb del módulo tensorflow.keras.datasets para cargar el conjunto de datos IMDB, CountVectorizer de sklearn.feature_extraction.text para preprocesar los datos de texto, LogisticRegression de sklearn.linear_model para crear y entrenar un modelo de regresión logística, y confusion_matrix, accuracy_score y log_loss de sklearn.metrics para evaluar el rendimiento del modelo.

```
import numpy as np
from tensorflow.keras.datasets import imdb
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, log_loss
```

Posteriormente, se carga el conjunto de datos IMDB utilizando la función load_data de imdb de tensorflow.keras.datasets. Se establece vocab_size en 10000 para limitar el tamaño del vocabulario. Los datos de entrenamiento y prueba se asignan a (x_train, y_train) y (x_test, y_test) respectivamente.

```
# Load the IMDB dataset with a vocabulary size of 10,000
vocab_size = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
```

Luego de ello se convierten los índices de palabras en texto legible. Se utiliza get_word_index() para obtener el diccionario de índices de palabras. Luego, se ajustan los índices para evitar colisiones con los índices reservados agregando 3 a cada valor del diccionario. Se agregan algunas palabras reservadas al diccionario. Luego, se crea un nuevo diccionario reverse_word_index donde las claves son los índices y los valores son las palabras correspondientes. Finalmente, se reemplazan los índices en x_train y x_test con las palabras correspondientes utilizando el diccionario reverse_word_index.

```
word_index = imdb.get_word_index()
word_index = {k: (v + 3) for k, v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2
word_index["<UNUSED>"] = 3
reverse_word_index = {v: k for k, v in word_index.items()}
x_train = [" ".join([reverse_word_index.get(i - 3, "<UNK>") for i in s]) for s in x_train]
x_test = [" ".join([reverse_word_index.get(i - 3, "<UNK>") for i in s]) for s in x_test]
```

Siguiente a ello, se utiliza CountVectorizer para preprocesar los datos de texto en representaciones de "bag-of-words" binarias. Se crea un objeto vectorizer con los parámetros binary=True para obtener una representación binaria y max_features=vocab_size para limitar el tamaño del vocabulario.

```
vectorizer = CountVectorizer(binary=True, max_features=vocab_size)
x_train_bbow = vectorizer.fit_transform(x_train)
x_test_bbow = vectorizer.transform(x_test)
```

Posteriormente, se declara el entrenamiento del modelo bajo el método de regresión logística con un máximo de 1000 iteraciones. Luego, se empieza a entrenar el modelo utilizando

los datos de entrenamiento (x_train_bbow) y las etiquetas de entrenamiento (y_train).

```
# Train a logistic regression model on the binary bag-of-words matrices
lr = LogisticRegression(max_iter=1000)
lr.fit(x_train_bbow, y_train)
```

Una vez entrenado el modelo, se pasa a su evaluación, estas líneas utilizan el modelo entrenado (lr.predict()) en los datos de entrenamiento (x_train_bbow). Luego, se calcula la pérdida (log_loss) y la precisión (accuracy_score) comparando las etiquetas reales (y_train) con las predicciones (train_pred) del conjunto de entrenamiento.

```
# Evaluate the logistic regression model on the binary bag-of-words matrices
train_pred = lr.predict(x_train_bbow)
train_loss = log_loss(y_train, train_pred)
train_acc = accuracy_score(y_train, train_pred)
```

Se realizan predicciones utilizando el modelo entrenado en los datos de prueba (x_test_bbow). Luego, se calcula la pérdida y la precisión comparando las etiquetas reales con las predicciones del conjunto de prueba.

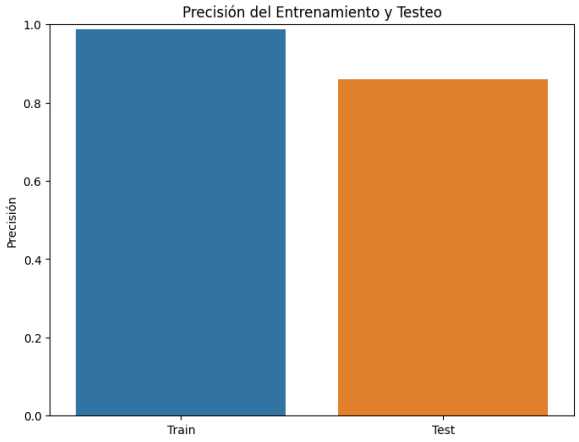
```
test_pred = lr.predict(x_test_bbow)
test_loss = log_loss(y_test, test_pred)
test_acc = accuracy_score(y_test, test_pred)
```

Finalmente se imprimen los resultados de la pérdida y la precisión del conjunto de entrenamiento y del conjunto de prueba. Los valores de pérdida representan qué tan bien se ajusta el modelo a los datos, mientras que la precisión indica qué tan acertadas son las predicciones del modelo en comparación con las etiquetas reales.

Se adjunta un gráfico de la precisión en ambos conjuntos de datos para mayor comprensión, concluyendo principalmente en la existencia de un indicio que prueba la ausencia de sobre entrenamiento del modelo, esto debido a que los datos de testeo no se ajustan perfectamente al conjunto de datos del entrenamiento.

```
print(f"Train loss: {train_loss}")
print(f"Train accuracy: {train_acc}")
print(f"Test loss: {test_loss}")
print(f"Test accuracy: {test_acc}")
```

```
Train loss: 0.42098987158488854
Train accuracy: 0.98832
Test loss: 5.021601790171802
Test accuracy: 0.86068
```



Se realizan predicciones utilizando el modelo entrenado en los conjuntos de entrenamiento y prueba. Las predicciones se almacenan en las variables y_pred_train y y_pred_test, respectivamente.

```
y_pred_train = lr.predict(x_train_bbow)
y_pred_test = lr.predict(x_test_bbow)
```

A continuación, se calcula la matriz de confusión utilizando las etiquetas reales (y_train y y_test) y las predicciones correspondientes (y_pred_train y y_pred_test). La matriz de confusión es una herramienta que muestra la cantidad de predicciones correctas e incorrectas para cada clase.

```
confusion_matrix_train = confusion_matrix(y_train, y_pred_train)
confusion_matrix_test = confusion_matrix(y_test, y_pred_test)
```

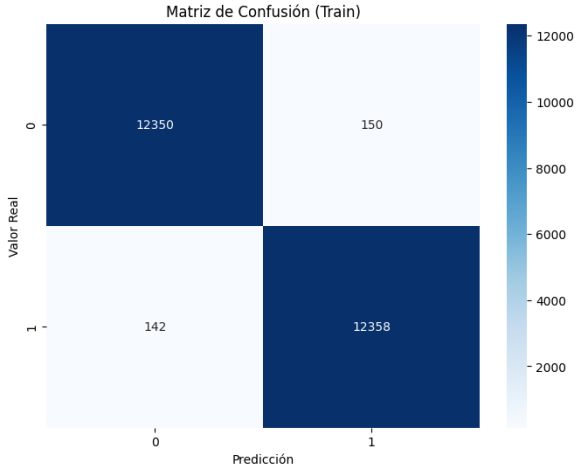
Finalmente, estas líneas imprimen en pantalla las matrices de confusión para el conjunto de entrenamiento y el conjunto de prueba. La matriz de confusión muestra la distribución de las predicciones en cada clase y proporciona información sobre los

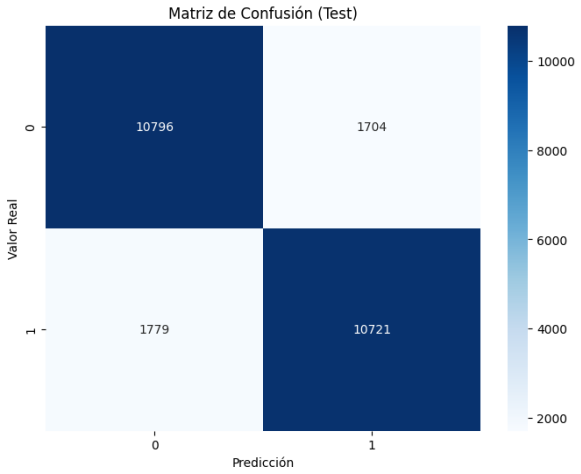
falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos.

```
print("Confusion matrix (Train):")
print(confusion_matrix_train)
print("Confusion matrix (Test):")
print(confusion_matrix_test)
```

```
Confusion matrix (Train):
[[12350   150]
 [  142 12358]]
Confusion matrix (Test):
[[10796  1704]
 [ 1779 10721]]
```

A manera de representación gráfica, se adjunta el resultado de la matriz de confusión para los datos de entrenamiento y los datos de testeo.





Tercer método: SVM con matrices TF-IDF

Se realiza a través del entrenamiento de un Support Vector Machine (SVM) en las matrices TF-IDF. Para comprender el contexto, SVM es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación y regresión. En el contexto del procesamiento de texto, SVM se aplica comúnmente para la clasificación de documentos o textos en categorías predefinidas. Su objetivo es encontrar un hiperplano óptimo que separe las diferentes clases de manera eficiente en un espacio de características de alta dimensión. La matriz TF-IDF se construye utilizando estas medidas para cada palabra en cada documento de la colección. Los valores de TF-IDF se calculan multiplicando la Frecuencia de Término por la Frecuencia Inversa de Documento.

En el contexto del procesamiento de texto, se utiliza el enfoque de SVM en combinación con la matriz TF-IDF para la clasificación de documentos. La matriz TF-IDF se utiliza como representación numérica de los documentos, donde cada palabra en el vocabulario se convierte en una característica con su respectivo valor de TF-IDF. Partiendo de esta base, a continuación, se describe el proceso llevado a cabo para

obtención de un modelo de red neuronal artificial a través de este método.

En primera medida, se importan las librerías requeridas para este desarrollo, junto al data set IMDB.

```
import numpy as np
from tensorflow.keras.datasets import imdb
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
import joblib

# Load the IMDB dataset with a vocabulary size of 10,000
vocab_size = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
```

Se convierten los índices de las palabras a texto, y se realiza un preprocesamiento de los datos en formato texto usando TF-IDF a través de un vectorizador.

```
# Convert the word indices to text
word_index = imdb.get_word_index()
word_index = {k: (v + 3) for k, v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2
word_index["<UNUSED>"] = 3
reverse_word_index = {v: k for k, v in word_index.items()}
x_train = [" ".join([reverse_word_index.get(i - 3, "<UNK>") for i in s]) for s in x_train]
x_test = [" ".join([reverse_word_index.get(i - 3, "<UNK>") for i in s]) for s in x_test]

# Pre-process the text data using TF-IDF
vectorizer = TfidfVectorizer(max_features=vocab_size)
x_train_tfidf = vectorizer.fit_transform(x_train)
x_test_tfidf = vectorizer.transform(x_test)
```

Entrenamos el SVM con svm.fit, luego evaluamos el modelo imprimiendo el accuracy.

```
# Train a support vector machine (SVM) on the TF-IDF matrices
svm = SVC()
svm.fit(x_train_tfidf, y_train)

# Evaluate the SVM model on the TF-IDF matrices
train_acc = svm.score(x_train_tfidf, y_train)
test_acc = svm.score(x_test_tfidf, y_test)

print(f"Train accuracy: {train_acc}")
print(f"Test accuracy: {test_acc}")
```

Se obtienen los resultados del entrenamiento.

```
Train accuracy: 0.98208
Test accuracy: 0.8854
```

A continuación, se guarda el vectorizador y el modelo en formato pkl

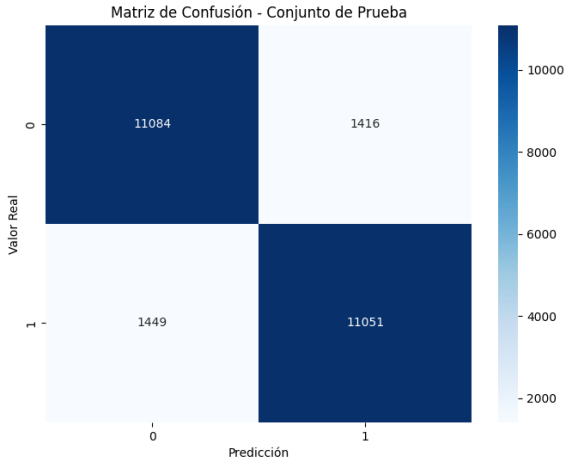
```
vectorizer = TfidfVectorizer()
vectorizer.fit(x_train)

# Guardar el vectorizador en un archivo
joblib.dump(vectorizer, "vectorizer.pkl")
```

```
['vectorizer.pkl']
```

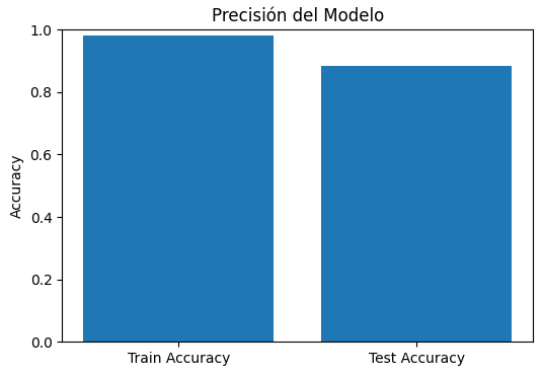
```
# Guardar el modelo en un archivo
joblib.dump(svm, 'modelo_svm.pkl')
#svm.save("modelo3.h5")
```

```
['modelo_svm.pkl']
```

La matriz de confusión arroja el siguiente resultado:



Gráficamente se puede observar la precisión del modelo, tanto para el entrenamiento como para el testeo.



Cuarto método: Padding y embedding

Se importan las bibliotecas necesarias, incluyendo NumPy, Keras.datasets y Keras.layers, para cargar y procesar los datos, y construir el modelo.

```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense
from tensorflow.keras.models import Sequential
```

Se carga el conjunto de datos de IMDB usando la función imdb.load_data(), se limita el tamaño del vocabulario a 10,000 para reducir la complejidad del modelo. Una forma de reducir la complejidad del modelo es limitar el tamaño del vocabulario, es decir, la cantidad de palabras únicas que se consideran en el modelo. significa que solo se considerarán las 10,000 palabras más frecuentes en el conjunto de datos, y las palabras menos frecuentes se descartarán. Aunque el conjunto de datos IMDB contiene 50,000 reseñas, limitar el tamaño del vocabulario a 10,000 palabras no necesariamente limita la cantidad de datos disponibles para entrenar el modelo. Las reseñas se representan como secuencias de índices de palabras, y cada índice representa una palabra única en el vocabulario. Si una reseña contiene una palabra que no está en el vocabulario limitado, esa palabra simplemente se ignora. Por lo tanto, el modelo aún tiene acceso a todas las reseñas, pero solo se consideran las palabras más comunes

```
# Load the IMDB dataset with a vocabulary size of 10,000
vocab_size = 10000
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)
```

Se establece la longitud máxima de la secuencia de entrada a 256 palabras, y se utiliza la función pad_sequences() de Keras para truncar o rellenar las secuencias de entrada para que todas tengan la misma longitud. **pad_sequenc**es() es una técnica utilizada en procesamiento de lenguaje natural (NLP), que se utiliza para transformar secuencias de diferentes longitudes en secuencias de una longitud fija. Es comúnmente utilizado en problemas de clasificación de texto, donde los modelos de aprendizaje automático requieren entradas de longitud fija.
El funcionamiento de pad_sequences() se puede describir en los siguientes pasos:

1. Se recibe una lista de secuencias de diferentes longitudes, donde cada secuencia es una lista de enteros. Cada entero representa una palabra o token en una oración.
2. Se define una longitud máxima para las secuencias. Las secuencias más cortas se rellenan con ceros en el comienzo o final para que tengan la misma longitud que las secuencias más largas.
3. Si la longitud de una secuencia es mayor que la longitud máxima definida, se trunca la secuencia para que tenga la misma longitud máxima.
4. La función devuelve una matriz de secuencias de longitud fija, donde cada fila representa una secuencia y cada columna representa una palabra en la secuencia. Si una secuencia es más corta que la longitud máxima, se rellena con ceros al principio o al final de la secuencia.

```
# Pad the sequences to a maximum length of 256 words
maxlen = 256
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)
```

Se construye el modelo de redes neuronales secuencial utilizando la API de Keras. El modelo consta de una capa de incrustación (Embedding) para convertir las secuencias de entrada en vectores de baja dimensionalidad, una capa de

reducción de dimensionalidad (GlobalAveragePooling1D) para reducir la complejidad del modelo y una capa densa (Dense) con una sola neurona de salida que utiliza la función de activación sigmoidal para predecir la polaridad de la crítica (positiva o negativa).
Para amplificar el proceso se da :

- La capa de incrustación (Embedding) se utiliza para convertir las secuencias de entrada en vectores de baja dimensionalidad. En este caso, se utiliza para representar las palabras en el vocabulario como vectores densos de longitud 128.
- La capa de reducción de dimensionalidad (GlobalAveragePooling1D) se utiliza para reducir la complejidad del modelo al realizar un promedio de los vectores de incrustación en cada secuencia. Esto permite que el modelo extraiga características relevantes de las secuencias de entrada de una manera más eficiente.
- La capa densa (Dense) con una sola neurona de salida utiliza la función de activación sigmoidal para predecir la polaridad de la crítica, es decir, si es positiva o negativa. La capa densa está conectada a la capa de reducción de dimensionalidad y realiza la tarea de clasificación binaria.

```
# Build a simple neural network model with an embedding layer
model = Sequential()
model.add(Embedding(vocab_size, 128, input_length=maxlen))
model.add(GlobalAveragePooling1D())
model.add(Dense(1, activation="sigmoid"))
model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Se entrena el modelo utilizando el método fit() de Keras. Se utilizan los datos de entrenamiento, un tamaño de lote de 32 y 10 épocas de entrenamiento. También se proporcionan los datos de validación para supervisar el rendimiento del modelo durante el entrenamiento.

```
# Train the neural network model on the word embedding sequences
history= model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test))
```

Se evalúa el rendimiento del modelo utilizando los datos de entrenamiento y de prueba. Se utiliza la función evaluate() de Keras para calcular la precisión (accuracy) del modelo.

```
# Evaluate the neural network model on the word embedding sequences
train_acc = model.evaluate(x_train, y_train)[1]
test_acc = model.evaluate(x_test, y_test)[1]
```

Se imprime la precisión del modelo en los datos de entrenamiento y de prueba y se guarda el modelo

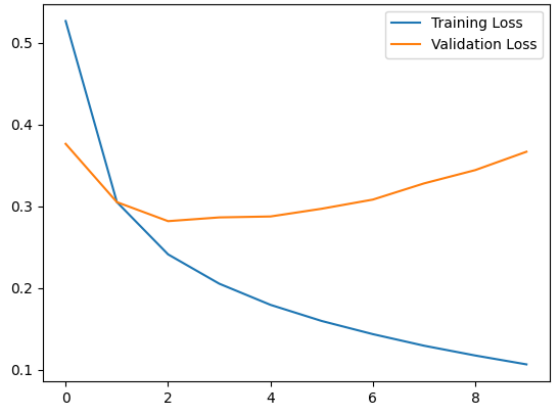
```
print(f"Train accuracy: {train_acc}")
print(f"Test accuracy: {test_acc}")

# Guardar el modelo entrenado
model.save("modelo.h5")
```

El output de la primera del código :

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 0s 0us/step
Epoch 1/10
782/782 [==============================] - 23s 27ms/step - loss: 0.5265 - accuracy: 0.7872 - val_loss: 0.3762 - val_accuracy: 0.8571
Epoch 2/10
782/782 [==============================] - 18s 23ms/step - loss: 0.3050 - accuracy: 0.8851 - val_loss: 0.3049 - val_accuracy: 0.8763
Epoch 3/10
782/782 [==============================] - 19s 24ms/step - loss: 0.2410 - accuracy: 0.9111 - val_loss: 0.2816 - val_accuracy: 0.8868
Epoch 4/10
782/782 [==============================] - 18s 23ms/step - loss: 0.2051 - accuracy: 0.9258 - val_loss: 0.2862 - val_accuracy: 0.8821
Epoch 5/10
782/782 [==============================] - 18s 23ms/step - loss: 0.1792 - accuracy: 0.9354 - val_loss: 0.2873 - val_accuracy: 0.8836
Epoch 6/10
782/782 [==============================] - 19s 24ms/step - loss: 0.1595 - accuracy: 0.9442 - val_loss: 0.2968 - val_accuracy: 0.8811
Epoch 7/10
782/782 [==============================] - 19s 24ms/step - loss: 0.1433 - accuracy: 0.9503 - val_loss: 0.3080 - val_accuracy: 0.8794
Epoch 8/10
782/782 [==============================] - 21s 27ms/step - loss: 0.1292 - accuracy: 0.9572 - val_loss: 0.3270 - val_accuracy: 0.8745
Epoch 9/10
782/782 [==============================] - 19s 24ms/step - loss: 0.1172 - accuracy: 0.9616 - val_loss: 0.3441 - val_accuracy: 0.8729
Epoch 10/10
782/782 [==============================] - 20s 25ms/step - loss: 0.1063 - accuracy: 0.9660 - val_loss: 0.3666 - val_accuracy: 0.8682
782/782 [==============================] - 2s 3ms/step - loss: 0.0881 - accuracy: 0.9756
782/782 [==============================] - 2s 3ms/step - loss: 0.3666 - accuracy: 0.8682
Train accuracy: 0.9755600094795227
Test accuracy: 0.8681600093841553
```

Se visualiza el Training loss y Validation loss y se obtiene la matriz de confusion

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix


# Obtener las predicciones del modelo en los datos de prueba
y_pred = model.predict(x_test)

# Convertir las probabilidades a etiquetas binarias
y_pred = np.round(y_pred)

# Obtener la matriz de confusión
cm = confusion_matrix(y_test, y_pred)

# Graficar la evolución del loss y accuracy durante el entrenamiento
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

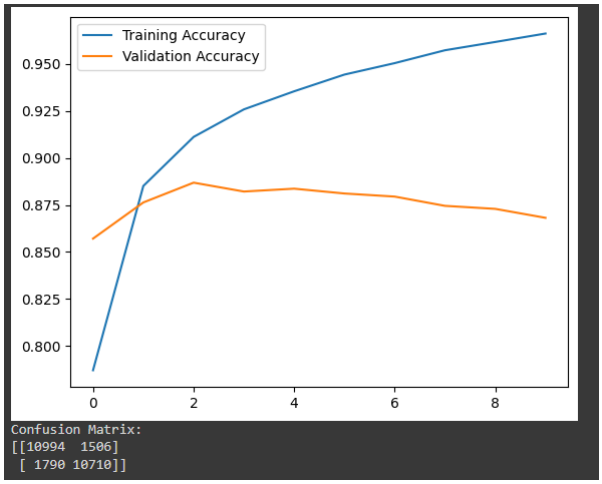


Se visualiza que el validation loss tiende a presentar overfitting por su forma exponencial y porque se puede identificar cuando

la pérdida de validación comienza a aumentar mientras que la pérdida de entrenamiento sigue disminuyendo.; Por lo que se debe de dar las configuraciones pertinentes para mejorar el modelo y que este sea preciso.

Se visualiza el Training y Validation Acurracy y se imprime la matriz de confusión :

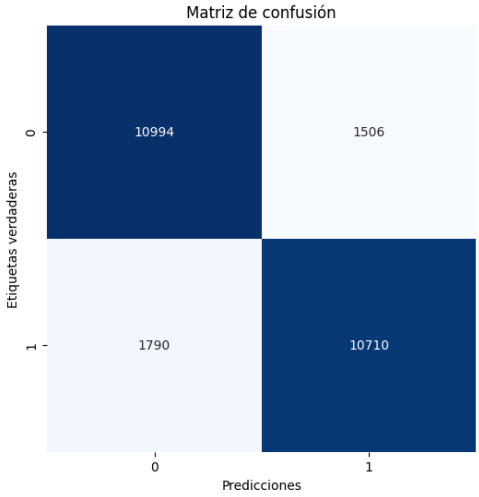
```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()

# Imprimir la matriz de confusión
print('Confusion Matrix:')
print(cm)
```



El accuray validation es proporcional al de entrenamiento debido a que no disminuye demasiado, dado esto para mejorar el modelo y que el accuracy mejore se debe tener encuentra el loss como anteriormente se mencionó presentaba overfiting por lo cual afecta al accuracy en su modo de comportarse.

Se visualiza la matriz de confusión, anteriormente se había imprimido, pero para una mejor visualización de elementos de la clasificación se realiza el siguiente código:

```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns


# Calcular la matriz de confusión
cm = confusion_matrix(y_test, y_pred)

# Plotear la matriz de confusión
plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de confusión')
plt.xlabel('Predicciones')
plt.ylabel('Etiquetas verdaderas')
plt.show()
```



La clase 1 representa la clase positiva que en este caso seria las reviews positivas y la clase 0 representa la clase negativa , que en este caso seria las reviews negativas, la clasificación las realiza de manera muy parecida . Depende de lo que se requiera de mayor prioridad como reviews positivas o negativas se tendra que disminuir los falsos positivos o negativos.

Para ver su funcionamiento se le pide al usuario que ingrese su review , después de eso se Tokeniza Cada palabra de la revisión se busca en el índice de palabras y se asigna el número entero correspondiente. Si una palabra no está presente en el índice, se asigna el valor 0. Los tokens se guardan en la lista user_tokens.

```
# Prompt the user to enter a review for classification
user_review = input("Enter a review for classification: ")

# Tokenize the user's review using the IMDB dataset's word index
word_index = imdb.get_word_index()
user_tokens = [word_index[word] if word in word_index else 0 for word in user_review.split()]
```

Los tokens del usuario se rellenan para tener una longitud máxima de 256 palabras utilizando la función pad_sequences(). Se crea una lista con los tokens del usuario y se pasa como argumento junto con el parámetro maxlen (definido previamente) para obtener una secuencia de longitud fija. El resultado se guarda en la variable user_padded. El modelo

entrenado se utiliza para predecir el sentimiento de la revisión del usuario utilizando la función predict(). Se pasa la secuencia de tokens del usuario (user_padded) como entrada al modelo y se obtiene una predicción de sentimiento. El resultado se guarda en la variable prediction. Finalmente, se imprime el sentimiento predicho basado en el umbral de 0.5. Si la predicción es igual o mayor que 0.5, se considera una revisión positiva. De lo contrario, se considera una revisión negativa.

```
# Pad the user's tokens to a maximum length of 256 words
user_padded = pad_sequences([user_tokens], maxlen=maxlen)

# Use the trained model to predict the sentiment of the user's review
prediction = model.predict(user_padded)[0][0]

# Print the predicted sentiment
if prediction >= 0.5:
    print(f"Prediction: {prediction}")
    print("Positive review!")
else:
    print(f"Prediction: {prediction}")
    print("Negative review.")
```

El output de ejemplo mismo del dataset seria ‘This show was an amazing, fresh & innovative idea in the 70's when’ en el cual

es review positiva y aplicando el modelo la clasificó correctamente como positiva.

```
Enter a review for classification: This show was an amazing, fresh & innovative idea in the 70's when
1/1 [==============================] - 0s 21ms/step
Prediction: 0.5279856324195862
Positive review!
```

Para solucionar le problema de overfiting y mejorrar el modelo se agrega librerías necesarias para realizarlo en el cual son Dropout , regularizers y EarlyStopping

```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping
```

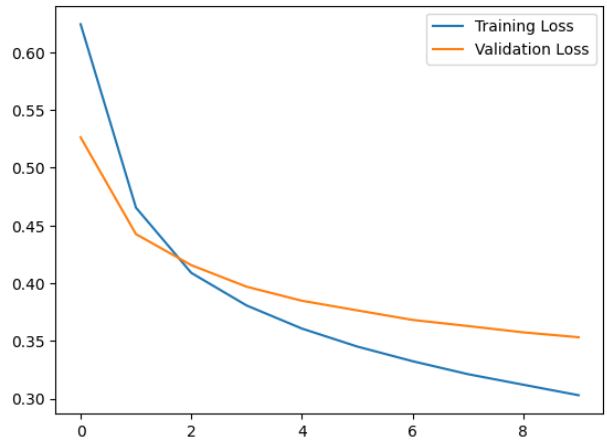
Y modificando el código de esta manera:

```
# Build a neural network model with an embedding layer, dropout, and regularization
model = Sequential()
model.add(Embedding(vocab_size, 128, input_length=maxlen))
model.add(GlobalAveragePooling1D())
model.add(Dropout(0.7))
model.add(Dense(1, activation="sigmoid", kernel_regularizer=regularizers.l2(0.001)))

model.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])

# Use early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

# Train the neural network model on the word embedding sequences
history = model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_test, y_test), callbacks=[early_stopping])
```

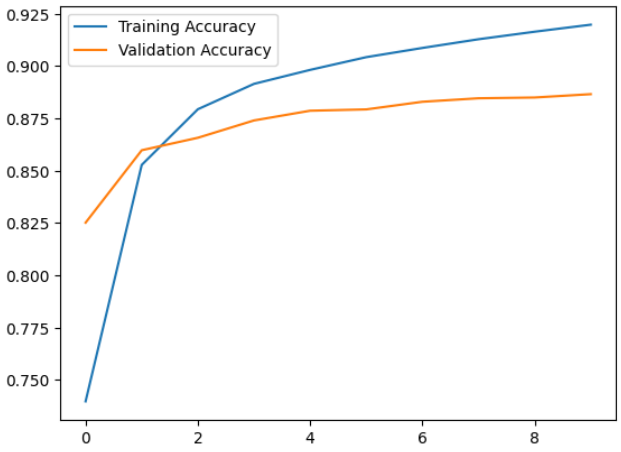
El ouput con la modificación del código mejoro en el acurracy y en el loss lo cual se evidencia que ya no tiene overfitting

```
Epoch 1/10
782/782 [==============================] - 25s 29ms/step - loss: 0.6242 - accuracy: 0.7398 - val_loss: 0.5263 - val_accuracy: 0.8252
Epoch 2/10
782/782 [==============================] - 21s 26ms/step - loss: 0.4654 - accuracy: 0.8528 - val_loss: 0.4425 - val_accuracy: 0.8598
Epoch 3/10
782/782 [==============================] - 23s 29ms/step - loss: 0.4089 - accuracy: 0.8794 - val_loss: 0.4156 - val_accuracy: 0.8657
Epoch 4/10
782/782 [==============================] - 20s 26ms/step - loss: 0.3807 - accuracy: 0.8915 - val_loss: 0.3969 - val_accuracy: 0.8740
Epoch 5/10
782/782 [==============================] - 22s 29ms/step - loss: 0.3607 - accuracy: 0.8982 - val_loss: 0.3848 - val_accuracy: 0.8786
Epoch 6/10
782/782 [==============================] - 22s 28ms/step - loss: 0.3451 - accuracy: 0.9047 - val_loss: 0.3764 - val_accuracy: 0.8791
Epoch 7/10
782/782 [==============================] - 21s 27ms/step - loss: 0.3324 - accuracy: 0.9086 - val_loss: 0.3682 - val_accuracy: 0.8829
Epoch 8/10
782/782 [==============================] - 21s 27ms/step - loss: 0.3212 - accuracy: 0.9128 - val_loss: 0.3629 - val_accuracy: 0.8846
Epoch 9/10
782/782 [==============================] - 22s 29ms/step - loss: 0.3121 - accuracy: 0.9164 - val_loss: 0.3574 - val_accuracy: 0.8858
Epoch 10/10
782/782 [==============================] - 24s 31ms/step - loss: 0.3030 - accuracy: 0.9198 - val_loss: 0.3532 - val_accuracy: 0.8866
782/782 [==============================] - 3s 3ms/step - loss: 0.2927 - accuracy: 0.9246
782/782 [==============================] - 2s 3ms/step - loss: 0.3532 - accuracy: 0.8866
Train accuracy: 0.9245600104331207
Test accuracy: 0.8865600228309631
```

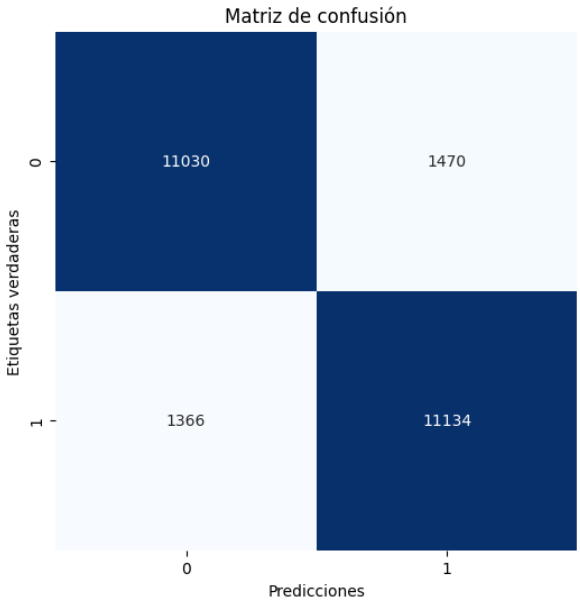
A continuación, se observa la gráfica de validation loss no está de forma exponencial y que se evita el overfiiting



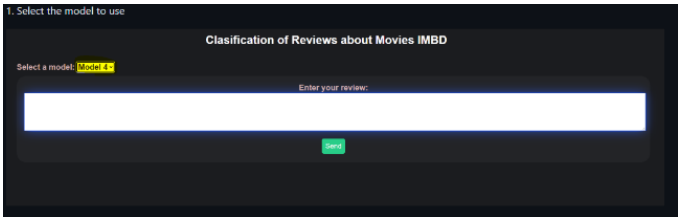
Y se visualiza la mejora del accuracy frente el otro modelo que tenía overfitting

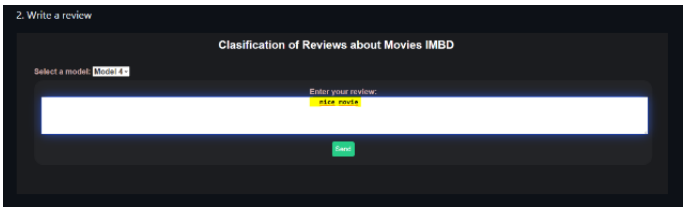


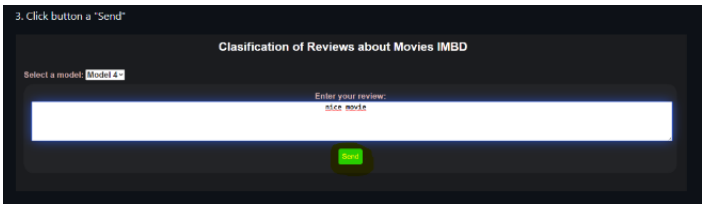
En la matriz de confusión se evidencia la mejora del modelo y que la modificación fue congruente con la clasificación de las clases

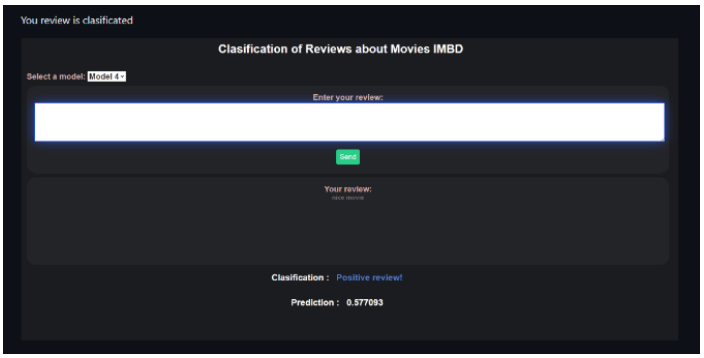


Con todos los modelos creados se realizó una aplicación con selección de los modelos para así realice la clasificación de la reviews en positivas o negativas en donde como prueba se realizó una corta review jamás entrenada por el modelo en el cual la review era “nice movie” Aplicacion en github (https://github.com/diegoperea20/Clasification-Reviews/tree/main/app):









III. RESULTADOS.

En cuanto al modelo 1, los resultados del accuracy son del 50%, lo cual indica que la predicción no se ajusta adecuadamente al modelo. Además, el validation loss muestra una gráfica exponencial, lo que sugiere un caso de overfitting.

De acuerdo al modelo 2, Si bien, el método de regresión logística puede representar una manera adecuada de realizar el modelo de clasificación que permita sortear satisfactoriamente la situación - problema presentada, no se logra evidenciar un

funcionamiento totalmente optimo del modelo entrenado en específico para poder filtrar información sensible de cara a los usuarios. Esto debido a la presencia de inconsistencias que se reflejan en una precisión de aproximadamente el 80%, la cual, debido a la robustez de su implementación puede ser adaptada y mejorada en el tiempo.

En cuanto al modelo 3, partiendo de los resultados obtenidos con Train accuracy: 0.98208 y Test accuracy: 0.8854, se puede afirmar que existe una alta eficiencia para realizar la clasificación con un porcentaje que se encuentra en un rango de alta aceptación dentro del campo de la inteligencia artificial si tenemos en cuenta la conclusión realizada por Kirsten Barkved en su artículo "Cómo saber si su modelo de aprendizaje automático tiene un buen rendimiento" [10]. El uso de SVM con las matrices TF-IDF fue óptimo, ya que estas técnicas son ampliamente utilizadas y han demostrado ser eficaces en problemas de clasificación de texto. Las matrices TF-IDF capturan la importancia relativa de las palabras en un documento, mientras que SVM es un algoritmo de aprendizaje supervisado que puede realizar una clasificación precisa incluso en espacios de alta dimensionalidad [11].

El modelo 4 presentaba overfitting con lo cual daba las clasificaciones de memoria por lo que las clasificaciones no eran precisas y no generaliza bien a nuevos datos y esto hace que sea difícil de interpretar su clasificación, por eso mejorando el código evitando el overfitting mejoró los resultados de clasificación, eso se evidencia en la matriz de confusión.

IV. CONCLUSIONES

Al tener diferentes enfoques y técnicas de preprocesamiento del conjunto de datos IMDB, así como variaciones en los modelos de entrenamiento, se pueden obtener diferentes perspectivas y resultados en la tarea de clasificación de sentimientos. La selección de la mejor solución depende de varios factores y requiere un análisis exhaustivo de los resultados obtenidos. Es importante explorar y comparar diferentes enfoques de preprocesamiento para determinar cuál funciona mejor para la tarea específica. Además, la elección del modelo de entrenamiento también tiene un impacto significativo en los resultados, por lo que es fundamental experimentar con diferentes modelos y ajustar sus hiperparámetros. En resumen, la variación en los enfoques de preprocesamiento y los modelos de entrenamiento permite encontrar la combinación óptima que maximiza el rendimiento en la clasificación de sentimientos.

Los diferentes modelos y alternativas de procesamiento y entrenamiento de los datos en este artículo han permitido dar a conocer las diferencias que se presentan a la hora de diseñar un modelo de machine learning, reflejando la subjetividad de la eficiencia según el contexto y aplicabilidad propuestos. En esta ocasión, para el caso inmediato de clasificación de sentimientos, ha resultado muy interesante el observar como un data set etiquetado y orientado hacia métodos de aprendizaje automáticos supervisados, ha podido beneficiarse ciertamente de ello, permitiendo obtener resultados decentes, con una precisión moderada y un margen de mejora elevado, en la clasificación que presenta la mayoría de los modelos planteados.

Como se pude evidenciar el método 1 se presenta un overfitting alto en comparación con otros métodos, una forma da llegar a un mejor resultado de este sería la reducción de la complejidad del modelo, si el modelo LSTM es demasiado complejo en relación con el conjunto de datos, existe la posibilidad de que esté memorizando los datos de entrenamiento en lugar de aprender patrones generales. Puedes considerar reducir el número de capas ocultas, disminuir la cantidad de unidades en cada capa o utilizar una arquitectura más simple en general.

REFERENCIAS

[1] "Tutorial Python: ¿Cómo combatir el Overfitting en el Machine Learning? | Codificando Bits". Codificando Bits. https://www.codificandobits.com/blog/tutorial-overfitting-machine-learning-python/ (accedido el 10 de mayo de 2023).

[2] "IMDB Dataset of 50K Movie Reviews". Kaggle: Your Machine Learning and Data Science Community. https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews (accedido el 10 de mayo de 2023).

[3] "Keras documentation: IMDB movie review sentiment classification dataset". Keras: Deep Learning for humans. https://keras.io/api/datasets/imdb/ (accedido el 10 de mayo de 2023).

[4] "Module: tf.keras.datasets.imdb | TensorFlow v2.12.0". TensorFlow. https://www.tensorflow.org/api_docs/python/tf/keras/datasets/imdb (accedido el 10 de mayo de 2023).

[5] "How to Build a Neural Network With Keras Using the IMDB Dataset". Built In. https://builtin.com/data-science/how-build-neural-network-keras (accedido el 10 de mayo de 2023).

[6] "| notebook.community". Jupyter Notebooks Gallery. https://notebook.community/aattaran/Machine-Learning-with-Python/Mini%20Project%20Student%20Admissions%20in%20Keras/imdb/IMDB_In_Keras (accedido el 10 de mayo de 2023).

[7]"tf.keras.datasets.imdb.load_data|TensorFlowv2.12.0".TensorFlow.
https://www.tensorflow.org/api_docs/python/tf/keras/datasets/imdb/load_data (accedido el 25 de mayo de 2023).

[8] “Build Your First Text Classifier in Python with Logistic Regression”. https://kavita-ganesan.com/news-classifier-with-logistic-regression-in-python/ (accedido el 25 de mayo de 2023).

[9] "How To Implement Logistic Regression Text Classification [2 Ways]". Spot Intelligence. https://spotintelligence.com/2023/02/22/logistic-regression-text-classification-python/ (accedido el 25 de mayo de 2023).

[10] How To Know if Your Machine Learning Model Has Good Performance | Obviously AI. (s. f.). https://www.obviously.ai/post/machine-learning-model-performance

[11] Bedi, G. (2020, 13 julio). A guide to Text Classification(NLP) using SVM and Naive Bayes with Python. Medium. https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34