Universidad Rafael Landívar
Facultad de Ingeniería
Inteligencia Artificial
Ing. Rolando Valdés

Proyecto
Predicción Tweets

Rodrigo Alejandro Villacinda Aguilar – 1205017
Catherine Lopex - 1055816

Ciudad de Guatemala, 25 de abril

Tabla de contenido

Introducción

El análisis de sentimientos mediante un texto nos permite identificar la emoción de un texto (positivo, negativo o neutral). En este proyecto se desarrolló un clasificador de sentimientos usando el algoritmo Naïve Bayes, aplicado a un conjunto de tweets. La solución fue implementada desde cero sin el uso de librerías externas para el modelo, y desplegada a través de una aplicación web.

Definición del problema y objetivos

Problema

determinar la emoción expresada en un tweet (positivo, negativo o neutral) en función de su contenido textual.

Objetivo General

Implementar un modelo de clasificación Naïve Bayes capaz de identificar el sentimiento (positivo, negativo o neutral) en tweets.

Objetivos específicos

- Realizar el preprocesamiento del dataset de tweets.
- Implementar el modelo Naïve Bayes desde cero.
- Evaluar el desempeño del modelo.
- Desarrollar una plataforma web para consultar el modelo.

Descripción de Dataset

Se utilizó un dataset (twets.csv) con tweets etiquetados en tres categorías: negativo, neutral, positivo. El dataset fue procesado para eliminar ruido como símbolos, URLs, stopwords, lematizado y vectorizado.

Descripción del Preporcesamiento Aplicado

Se realizo la limpieza y tratamiento del dataset tweets.csv siguiendo el siguiente pipeline utilizado JupyterNotebook:

1. Limpieza de texto: en este punto se realizo una conversión a minúsculas, eliminación de signos de puntuación, menciones y URLs, almacenados en un dataset modificado.
2. Tokenización: se separaron las palabras del texto limpio del proceso anterior. Almacenados en un dataset modificado.
3. Lematización: se estandarizaron las palabras buscando su forma raíz para un mejor tratamiento. Almacenados en un dataset modificado.
4. Vectorización: se utiliza la libreriza BOW de Python para vectorizar las plabras encontradas en los proceso anteriores, estas palabras son las que dan valor, adicional se eliminan palabras como the, this, etc.

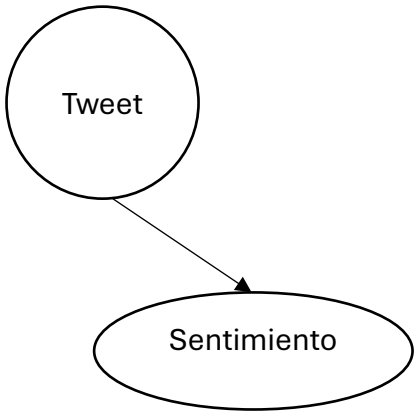
Se realizo la codificación del algoritmo Naïve Bayes:

El modelo se implementó utlizando la fórmula clásica de Naïve Bayes, fórmula:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

X: conjunto de palabras (tweet)
C: clase (positivo, neutral, negativo)

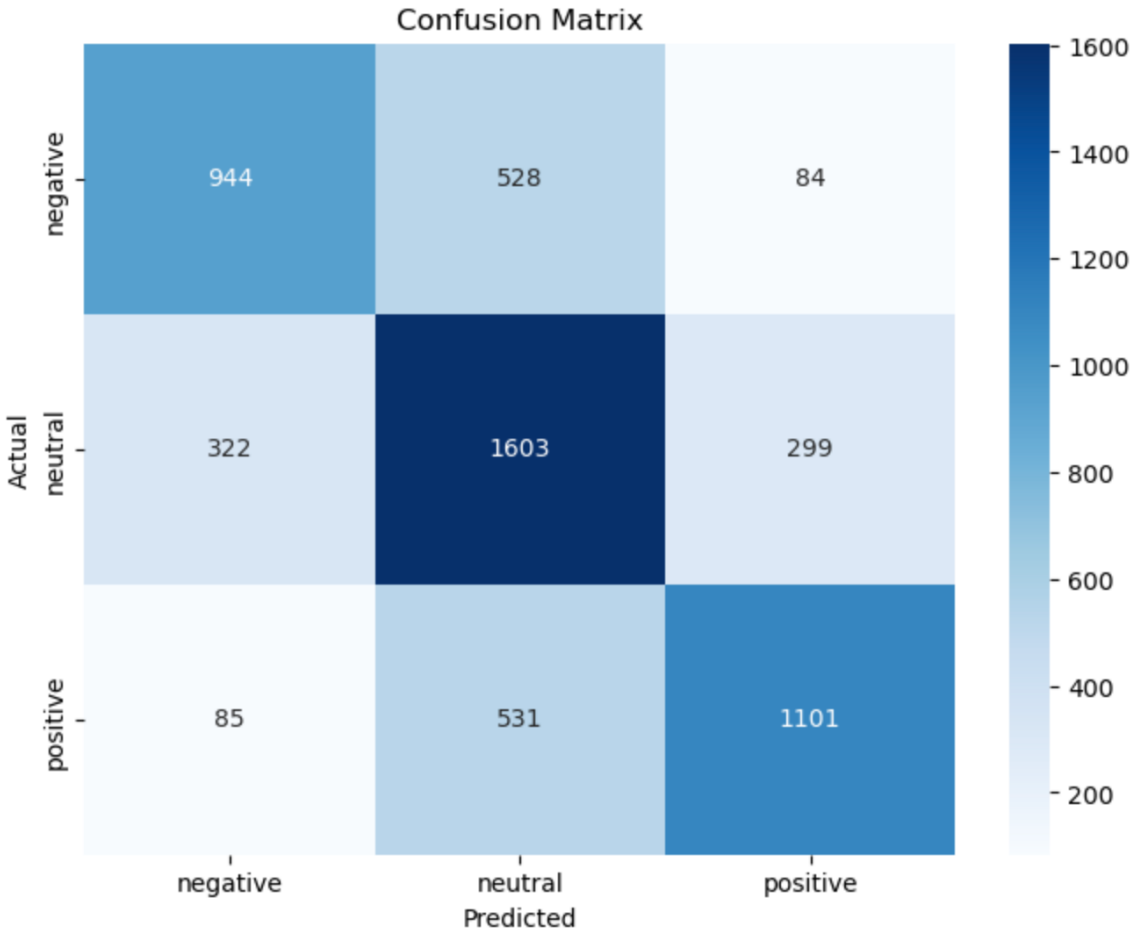
Red Bayesiana:



Explicación de la Evaluación del Modelo

Se evaluó el modelo mediante:

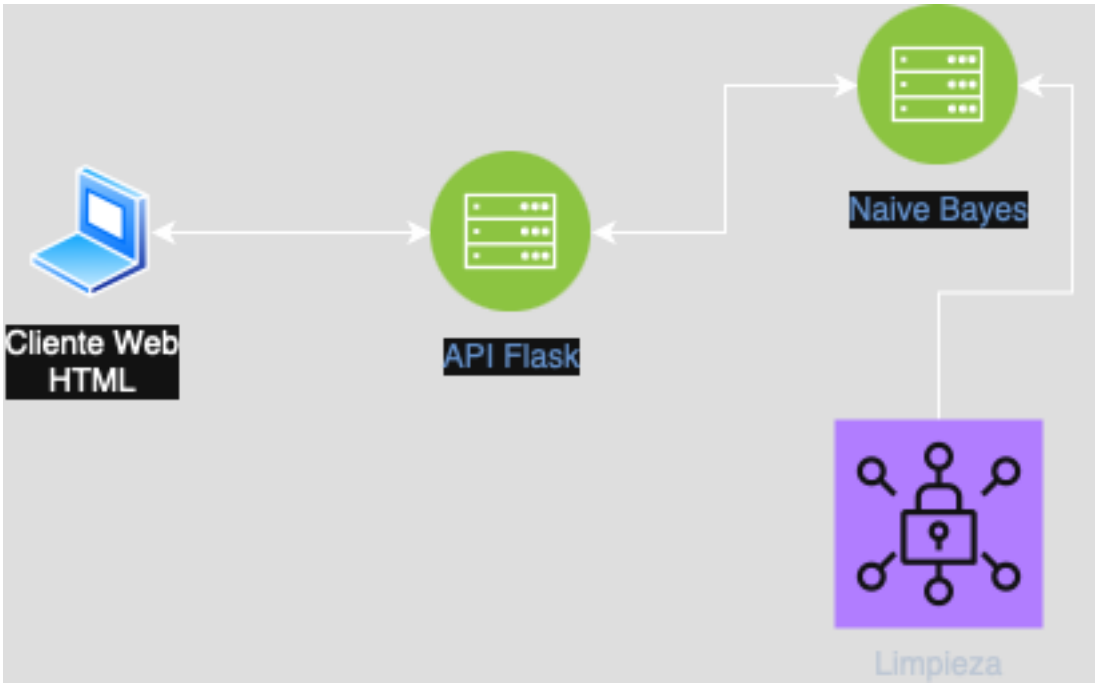
- **Matriz de confusión:**
  Mostró una fuerte confusión entre las clases "negative" y "neutral".



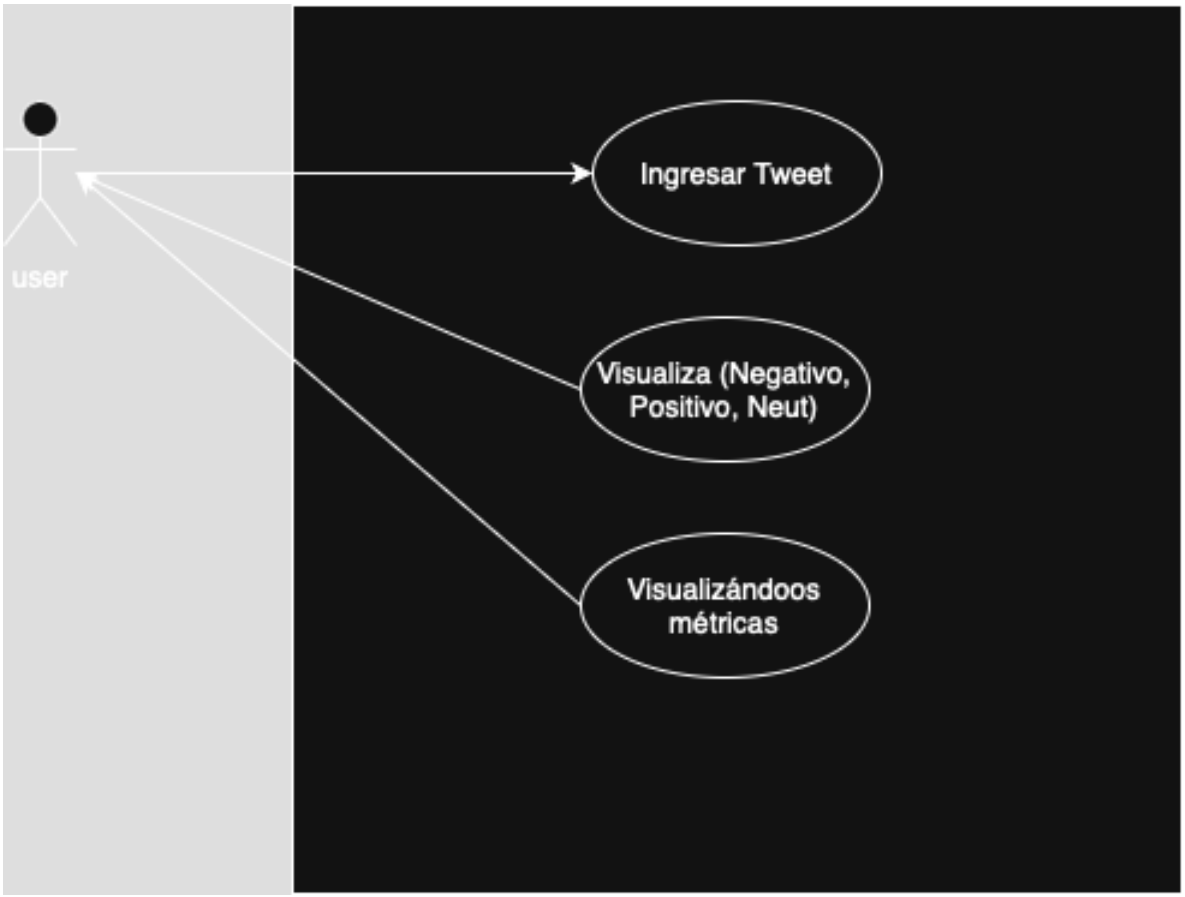
- **Métricas por clase:**
  - Clase "Positive": mejor desempeño (Precision = 0.70, Recall = 0.61).
  - Clase "neutral": resultados bajos.
- **Promedio Macro:**
  - Precision: 0.68
  - Recall: 0.66
  - F1-Score: 0.66

📉 Esto indica que el modelo puede mejorar en precisión y balance entre clases.

Arquitectura solución:



Casos de uso:



Flujo general:

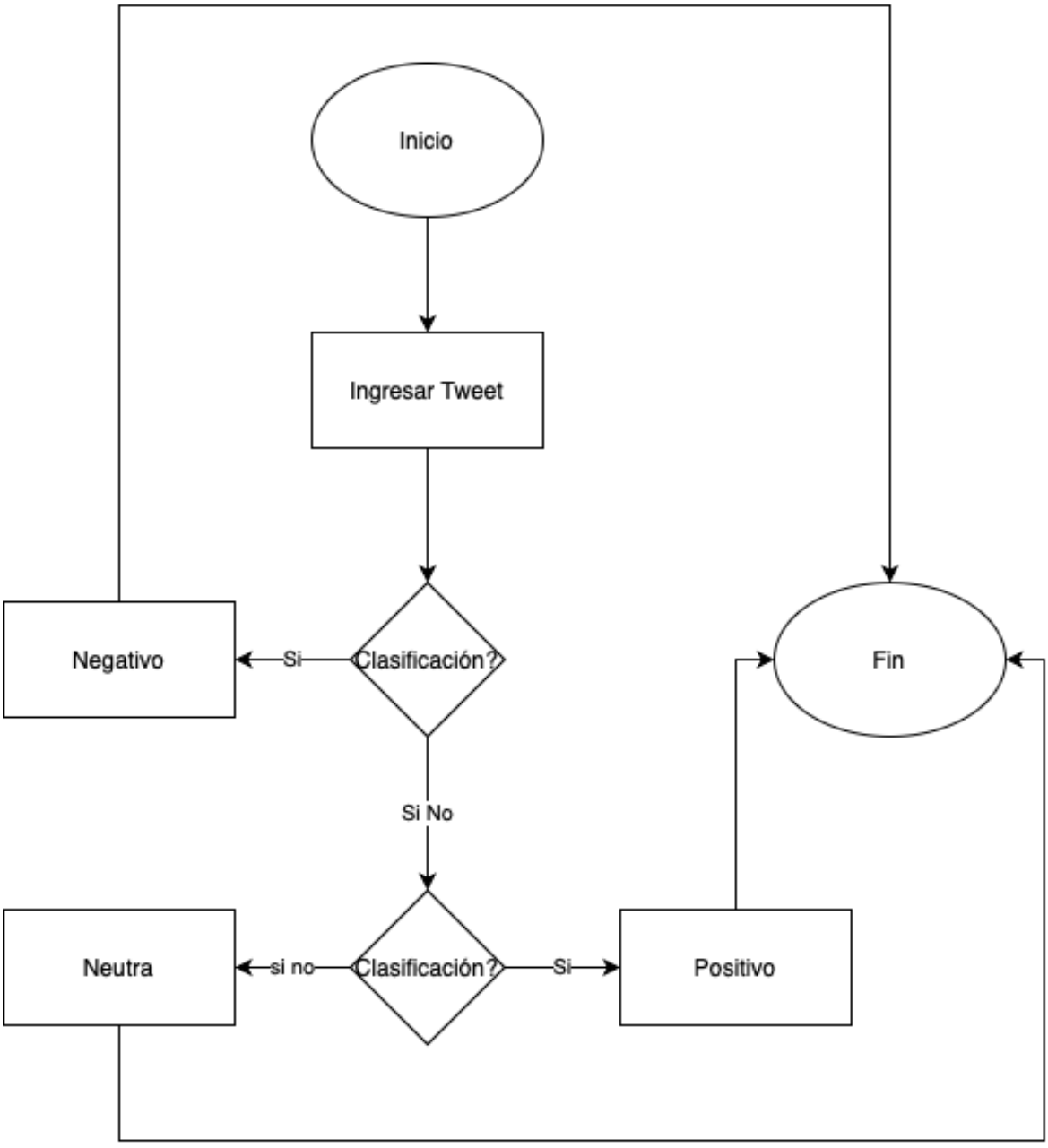
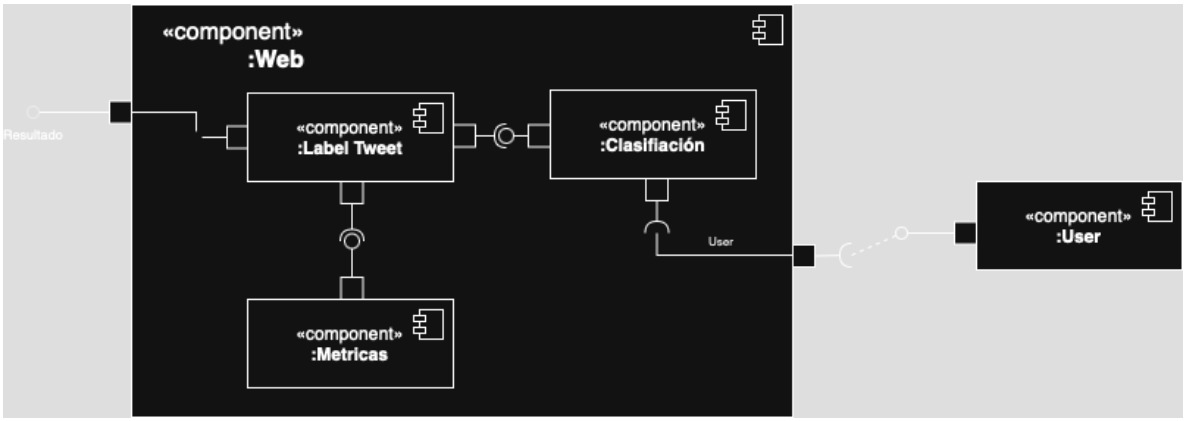
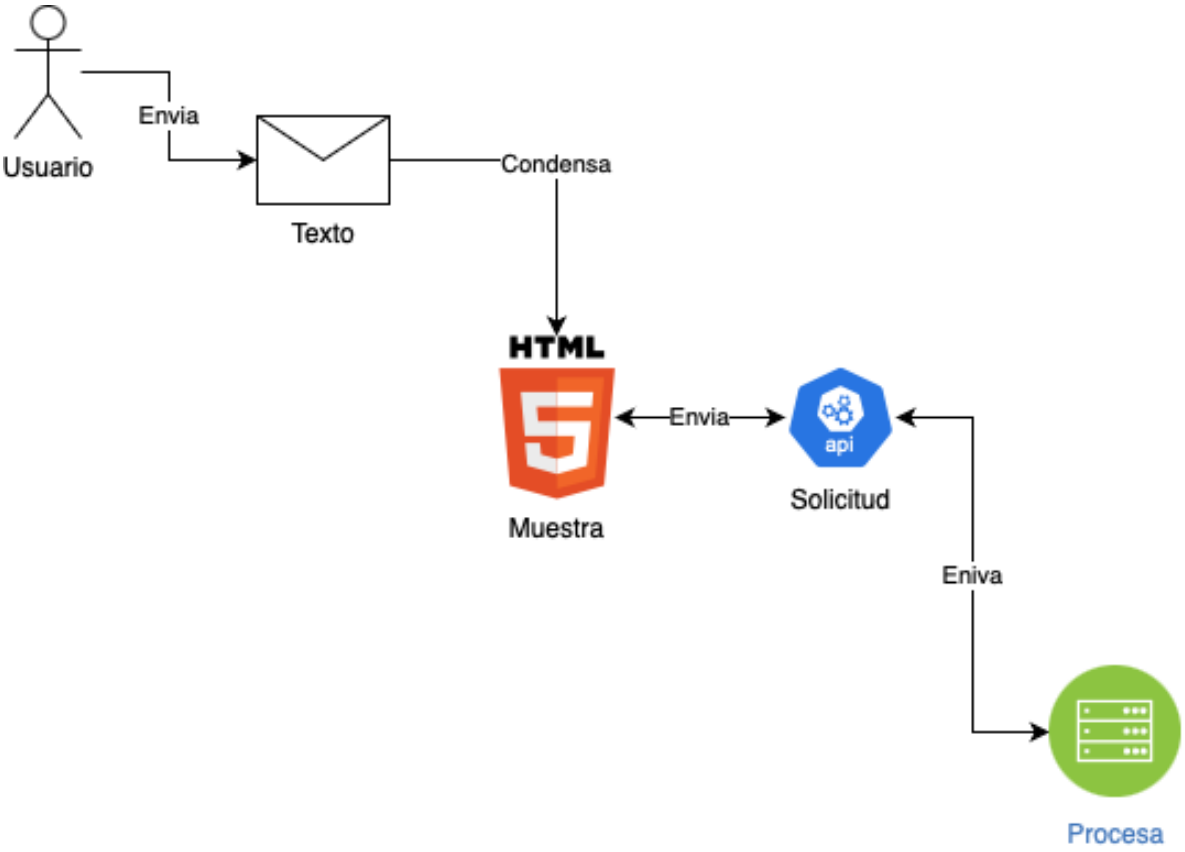


Diagrama de componentes:



Secuencia de interacción:



Evidencias de Funcionamiento

Código Modelo – Entrenamiento/API:

```
from flask import Flask, jsonify, request
import pandas as pd
import numpy as np
import re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

from flask_cors import CORS
app = Flask(__name__)
CORS(app)

class NaiveBayes:
  def fit(self, X, y):
    self.classes = np.unique(y)
    self.class_probs = {}
    self.word_probs = {}
    for c in self.classes:
      X_c = X[y == c]
      self.class_probs[c] = X_c.shape[0] / X.shape[0]
      total_wc = X_c.sum()
      self.word_probs[c] = (X_c.sum(axis=0) + 1) / (total_wc + X.shape[1])

  def predict(self, X):
    predictions = []
    for i in range(X.shape[0]):
      posteriors = {}
      row = X[i].toarray()[0]
      for c in self.classes:
        log_prob = np.log(self.class_probs[c]) + np.sum(row *
np.log(self.word_probs[c].A1))
        posteriors[c] = log_prob
      predictions.append(max(posteriors, key=posteriors.get))
    return np.array(predictions)

# Entrenamiento al inicio para mantener vectorizador y modelo en memoria
df = pd.read_csv("03lemmatized.csv")
df = df.dropna(subset=["text_lema", "sentiment"])
df = df[df["text_lema"].str.strip() != ""]
```

```
vectorizer = CountVectorizer(stop_words='english', min_df=5, max_df=0.8,
max_features=3000)
X = vectorizer.fit_transform(df["text_lema"])
y = df["sentiment"].astype(str)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

model = NaiveBayes()
model.fit(X_train, y_train)

@app.route('/metrics', methods=['GET'])
def metrics():
  y_pred = model.predict(X_test)
  accuracy = (y_pred == y_test.values).mean()
  report = classification_report(y_test, y_pred, output_dict=True)
  conf_matrix = confusion_matrix(y_test, y_pred).tolist()

  return jsonify({
    "accuracy": round(accuracy, 4),
    "classification_report": report,
    "confusion_matrix": conf_matrix
  })

@app.route('/predict', methods=['POST'])
def predict():
  data = request.get_json()
  tweet = data.get("text", "")

  # Preprocesamiento básico como en entrenamiento
  tweet = tweet.lower()
  tweet = re.sub(r"http\S+|www\S+", "", tweet)
  tweet = re.sub(r"@\w+|#\w+", "", tweet)
  tweet = re.sub(r"[^a-z\s]", "", tweet)
  tweet = re.sub(r"\s+", " ", tweet).strip()

  tweet_vec = vectorizer.transform([tweet])
  prediccion = model.predict(tweet_vec)[0]

  return jsonify({"prediccion": prediccion})

if __name__ == '__main__':
  app.run(debug=True)
```

Consola de métricas:

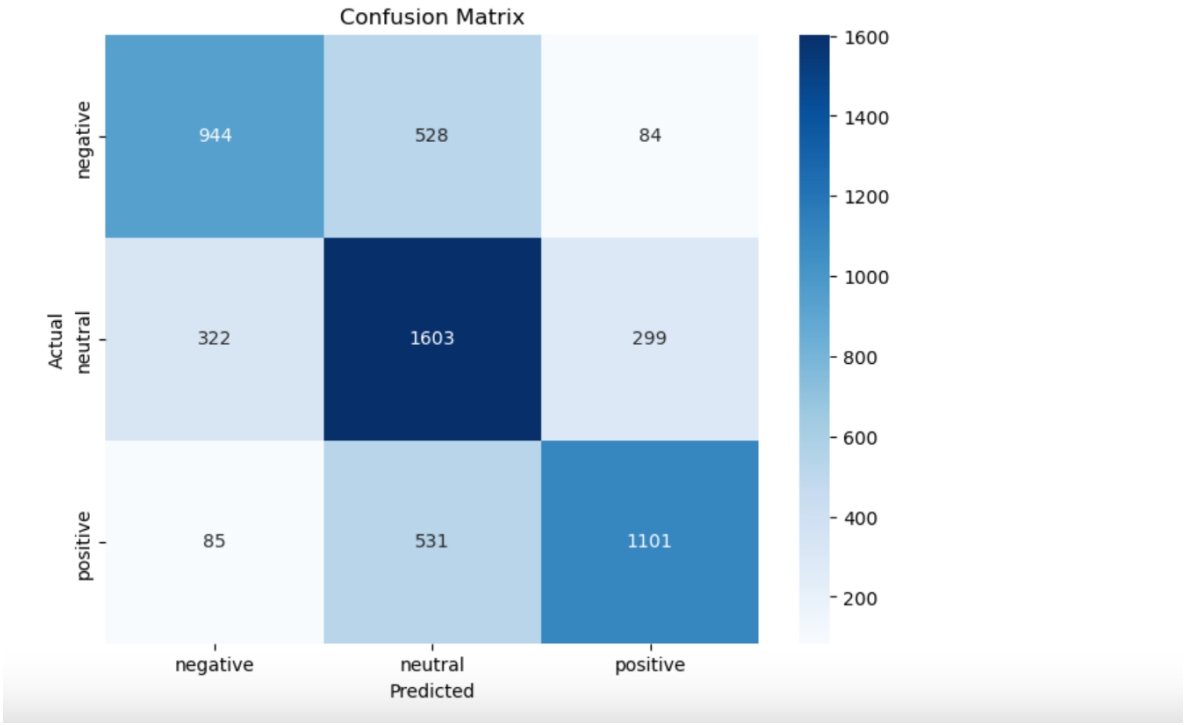
```
✅ Accuracy: 0.6636

📊 Matriz de Confusión:
[[ 944  528   84]
 [ 322 1603  299]
 [  85  531 1101]]

📈 Reporte de Clasificación:
              precision    recall  f1-score   support

    negative       0.70      0.61      0.65      1556
     neutral       0.60      0.72      0.66      2224
    positive       0.74      0.64      0.69      1717

    accuracy                           0.66      5497
   macro avg       0.68      0.66      0.66      5497
weighted avg       0.67      0.66      0.66      5497
```



Interfaz Web

Conclusiones

- El modelo Naïve Bayes es una solución simple pero útil para tareas de clasificación de texto.
- La mayor dificultad estuvo en el preprocesamiento de texto, especialmente en la limpieza y lematización.
- Es necesario balancear mejor el dataset o aplicar técnicas como SMOTE o ajuste de prior probabilities para mejorar los resultados en clases minoritarias.