UNIVERSIDAD BERNARDO O'HIGGINS

Facultad de Ingeniería, Ciencias y Tecnología

Clasificación de Phishing de correos electrónicos mediante Regresión Logística

Integrantes: Javier Ramírez, Bastian Vejar y Sergio Ruiz.

Docente: Enrique Gonzalo Pastene Aceituno

Fecha: 13-11-2025

Curso: SISTEM. INTELIG. [TEO 1]

UNIVERSIDAD BERNARDO O'HIGGINS

Índice

Contenido

UNIVERSIDAD BERNARDO O'HIGGINS

1. Descripción del Problema

El uso del correo electrónico ha aumentado significativamente en los últimos años, lo que también ha generado un incremento en la cantidad de mensajes maliciosos, especialmente aquellos relacionados con spam y phishing. Estos correos buscan engañar a los usuarios para obtener información sensible o instalar software dañino sin que estos lo adviertan. Dado que actualmente se reciben cientos de mensajes a diario, revisarlos manualmente ya no es una opción realista ni eficiente.

En este trabajo, nuestro grupo analiza este problema y propone un método para clasificar automáticamente los correos como spam o no spam. Para ello, se emplean técnicas de aprendizaje supervisado aplicadas directamente al contenido del texto. El objetivo es facilitar la detección temprana de correos potencialmente peligrosos y mejorar la seguridad del usuario sin necesidad de una revisión manual constante.

1.1 Objetivo general

Desarrollar un modelo de Machine Learning que nos permita identificar correos maliciosos a partir del contenido del asunto y el cuerpo del mensaje.

1.2 Objetivos específicos

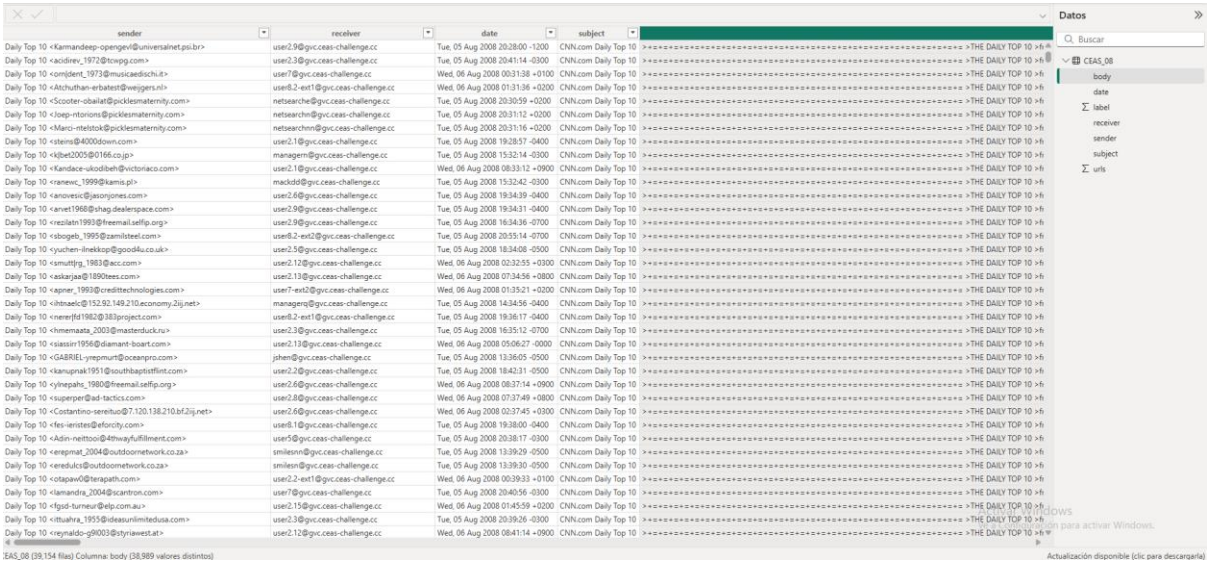
- Analizar y preprocesar el texto para convertirlo en datos útiles.
- Entrenar un modelo de clasificación binaria utilizando Regresión Logística.
- Evaluar el desempeño del modelo mediante métricas como accuracy, precision, recall y AUC.
- Determinar la efectividad del enfoque TF-IDF + Regresión Logística para la detección automática de spam.

2. Descripción del Dataset

El dataset utilizado corresponde al archivo **CEAS_08.csv**, el cual contiene correos reales etiquetados como spam (1) o legítimos (0). Este dataset es parte de un conjunto utilizado frecuentemente en tareas de clasificación de correo electrónico.

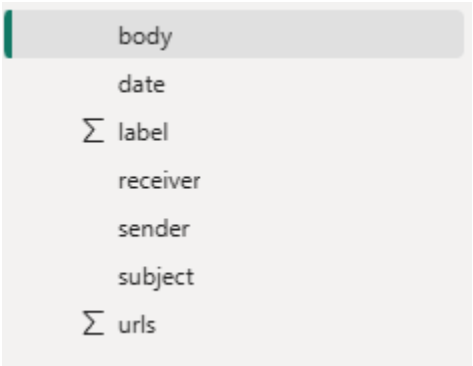
2.1 Características del dataset

- **Número total de registros:** 39.154
- **Correos spam:** 21.842
- **Correos legítimos:** 17.312
- **Número de variables:** 7



2.2 Columnas principales

- **sender:** remitente del correo
- **receiver:** destinatario
- **date:** fecha del mensaje
- **subject:** asunto del correo
- **body:** contenido completo del mensaje
- **urls:** cantidad de URLs presentes
- **label:** etiqueta (0 = legítimo, 1 = spam)



2.3 Limpieza y transformaciones

Aplicamos los siguientes pasos:

- Conversión a minúsculas
- Eliminación de URLs
- Eliminación de caracteres especiales
- Eliminación de números
- Eliminación de stopwords
- Unión del texto del *subject* y del *body* en una columna única llamada **text**

Estas transformaciones permiten que el modelo se enfoque en las palabras que realmente aportan información para diferenciar un correo spam de uno legítimo. En los correos maliciosos suelen aparecer términos asociados a ofertas engañosas, urgencias o promesas poco realistas, como “free”, “offer”, “credit”, “win”, “urgent”, “discount”, “money”, “click here” o “limited time”. Al limpiar y normalizar el texto, estas palabras relevantes se vuelven más fáciles de identificar para el modelo, mientras que elementos irrelevantes como números, URLs o símbolos dejan de interferir en el proceso de clasificación. De esta manera, se mejora la calidad del texto utilizado y se facilita que el algoritmo aprende patrones reales del lenguaje típico de los correos spam.

3. Justificación del algoritmo

El algoritmo que seleccionamos para este proyecto fue Regresión Logística, principalmente porque es uno de los modelos más adecuados para resolver problemas de clasificación binaria, cómo distinguir entre correos legítimos y correos de tipo phishing o spam. Consideramos que este modelo era una buena opción porque funciona especialmente bien cuando los datos han sido transformados mediante técnicas como TF-IDF, ya que permite identificar relaciones lineales entre ciertas palabras y la probabilidad de que un correo sea malicioso.

Otra razón por la que elegimos este modelo es que la Regresión Logística es interpretable, lo que nos permitió revisar sus coeficientes y entender qué términos del texto influían más en la clasificación. Esto nos ayudó a validar que el modelo estaba aprendiendo patrones reales presentes en el dataset y no simplemente memorizando los datos.

Además, la Regresión Logística es un algoritmo rápido, estable y eficiente, lo que facilitó su entrenamiento incluso con miles de características generadas por TF-IDF. Su mecanismo de regularización también ayudó a evitar el sobreajuste, mejorando su capacidad para generalizar frente a correos que no estaban en el entrenamiento.

En cuanto a la representación del texto, optamos por utilizar TF-IDF porque transforma el contenido de cada correo en valores numéricos que expresan la relevancia de cada palabra dentro del mensaje. Esto permitió que el modelo otorgara mayor importancia a términos característicos del spam y redujera el peso de palabras demasiado comunes. En conjunto, la combinación de TF-IDF y Regresión Logística nos entregó una solución sólida, eficiente y altamente efectiva para este tipo de problemas de clasificación de texto.

4. Descripción detallada del entrenamiento

A continuación presentamos el código que utilizamos en el desarrollo de nuestro modelo, organizado en distintos bloques para facilitar su comprensión. Cada parte del código cumple una función específica dentro del proceso de preparación, entrenamiento y evaluación del clasificador. Después de cada imagen incluimos un párrafo breve donde explicamos de forma clara el propósito de ese bloque y su importancia dentro del flujo completo del proyecto.

4.1 Importación de librerías

```
# 1. Importamos las librerías
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report,
    roc_curve,
    auc
)

import re
import string
import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')
stopwords = stopwords.words("english")
```

En esta sección importamos las librerías necesarias para realizar el análisis y el modelamiento del proyecto. Utilizamos *pandas* y *numpy* para manipular los datos, mientras que *matplotlib* y *seaborn* nos permitieron generar gráficos y visualizaciones. Desde *scikit-learn* incorporamos herramientas esenciales como la división del dataset, la vectorización del texto mediante TF-IDF, el modelo de Regresión Logística y las métricas de evaluación. Además, empleamos *re* y *nltk* para limpiar el texto y eliminar stopwords, dejando el contenido de los correos preparado para las etapas posteriores del procesamiento.

4.2 Se carga el dataset

```
# 2. Cargamos el dataset
from google.colab import files

# Reemplazamos nuestro dataset de phishing
df = pd.read_csv('/content/CEAS_08.csv')
df.head()
```

	sender	receiver
0	Young Esposito <Young@iworld.de>	user4@gvc.ceas-challenge.cc
1	Mok <ipline's1983@icable.ph>	user2.2@gvc.ceas-challenge.cc
2	Daily Top 10 <Karmandeep-opengevl@universalnet...	user2.9@gvc.ceas-challenge.cc
3	Michael Parker <ivqrnai@pobox.com>	SpamAssassin Dev <xrh@spamassassin.apache.org>
4	Gretchen Suggs <externalsep1@loanofficertool.com>	user2.2@gvc.ceas-challenge.cc

En este bloque realizamos la carga del archivo *CEAS_08.csv*, que contiene los correos utilizados en el proyecto. Usando `pandas.read_csv()`, importamos los datos y se almacenan en un DataFrame para poder analizarlos y procesarlos. Con `df.head()` mostramos las primeras filas del dataset, lo que permite verificar visualmente que el archivo se cargó correctamente y confirmar la estructura de las columnas, como el remitente, destinatario, asunto y cuerpo del correo. Esta revisión inicial es importante para asegurar que los datos están completos y listos para comenzar el preprocesamiento.

4.3 Visualización previa de los datos

```
# 3. Imprimimos las filas, columnas y luego con info a traves de la etiqueta mostramos la cantidad de correos legitimos y spam.
print("Filas, Columnas:", df.shape)
df.info()
df["label"].value_counts()
```

```
Filas, Columnas: (39154, 7)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39154 entries, 0 to 39153
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   sender    39154 non-null  object
 1   receiver  38692 non-null  object
 2   date      39154 non-null  object
 3   subject   39126 non-null  object
 4   body      39154 non-null  object
 5   label     39154 non-null  int64 
 6   urls      39154 non-null  int64 
dtypes: int64(2), object(5)
memory usage: 2.1+ MB
```

label	count
1	21842
0	17312

dtype: int64

En este bloque revisamos la estructura general del dataset para entender cuántas filas y columnas tiene, qué tipo de información contiene y si existen datos faltantes. También mostramos cuántos correos son spam y cuántos son legítimos. Esta revisión es importante porque permite confirmar que el archivo está completo y que la distribución entre las dos clases es adecuada para entrenar un modelo de clasificación. En este caso, el dataset tiene 39.154 correos, de los cuales 21.842 son spam y 17.312 son legítimos, lo que representa una proporción equilibrada para trabajar.

4.4 Limpieza básica de datos

```python
def clean_text(text):

    if isinstance(text, str) == False:
        text = str(text)

    # Convertimos todo a minúsculas
    text = text.lower()

    # Quitamos los URL del dataset
    text = re.sub(r"http\S+|www\S+|https\S+", "", text)

    # Quitamos simbolos y caracteres especiales
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)

    # Quitamos los números
    text = re.sub(r"\d+", "", text)

    # Eliminamos las conjugaciones y preposiciones
    words = [word for word in text.split() if word not in stopwords]
    text = " ".join(words)

    return text
```

En este bloque definimos una función encargada de limpiar el texto de cada correo antes de procesarlo. Primero convertimos todo a minúsculas para unificar el formato, luego eliminamos las URLs, símbolos, caracteres especiales y números, ya que no aportan información útil para identificar spam. Finalmente, removemos stopwords, que son palabras muy comunes como “the”, “and” o “is”, para que el modelo pueda enfocarse en los términos realmente relevantes. Esta limpieza es fundamental para reducir ruido y dejar únicamente el contenido que ayudará al modelo a aprender patrones propios de los correos spam.

4.5 Limpieza y unión de texto

```
# limpiamos los asuntos de los correos
df["subject"] = df["subject"].astype(str).apply(clean_text)
# Limpiamos el contenido del correo
df["body"] = df["body"].astype(str).apply(clean_text)
# Unimos ambos textos en una sola columna llamada
df["unión de texto"] = df["subject"] + " " + df["body"]

df["unión de texto"].head()
```

	unión de texto
0	never agree loser buck troubles caused small d...
1	befriend jenna jameson upgrade sex pleasures t...
2	cnncom daily top daily top cnncom top videos s...
3	svn commit r spamassassintrunk libmailspamassa...
4	specialpricespharmmoreinfo welcomefastshipping...

dtype: object

En este bloque aplicamos la función de limpieza tanto al asunto (*subject*) como al cuerpo (*body*) de cada correo, asegurando que ambas partes del mensaje queden preparadas para el análisis. Luego, los dos textos se combinan en una sola columna llamada **“unión de texto”**, lo que permite trabajar con un único contenido completo por correo. Esto es importante porque muchas señales de spam pueden aparecer tanto en el asunto como en el cuerpo, y unirlos ayuda al modelo a tener una visión más completa del mensaje para identificar patrones relevantes.

4.6 Vectorización

```
# Vectorizamos el texto
# TF = frecuencia de una palabra en un correo
# IDF = qué tan rara es esa palabra en el dataset
# ax_features=1000 = usamos las 1000 palabras mas "relevantes"
tfidf = TfidfVectorizer(max_features=1000)
# Con esto ayudamos a evitar el overfitting y producimos un mejor modelo
X = tfidf.fit_transform(df["unión de texto"])
y = df["label"]
```

En este bloque se transforma el texto limpio en valores numéricos mediante TF-IDF, una técnica que mide qué tan importante es cada palabra dentro de los correos. Se utilizó un límite de 1000 características para evitar sobreajuste y enfocarse solo en las palabras más relevantes. El resultado es una matriz donde cada correo queda representado por números en lugar de texto, lo que permite al modelo de Machine Learning interpretar el contenido y aprender patrones de clasificación entre spam y correos legítimos.

4.7 Entrenamiento y modelo

```
# Aca a traves del rtest-size le decimos que un 20% sea de prueba y lo demas que sea de entrenamiento
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
```

En este bloque separamos los datos en dos conjuntos: un 80% para entrenar el modelo y un 20% para evaluarlo. Esto nos permite comprobar si el modelo es capaz de generalizar y detectar spam en correos que nunca ha visto antes. Además, utilizamos *stratify=y* para mantener la misma proporción de spam y correos legítimos en ambos conjuntos, asegurando una evaluación más justa y representativa.

4.8 Entrenamiento con regresión logística

```
# Entrenar modelo
# el C previene sobreajuste y el max iter bace que el modelo converge
model = LogisticRegression(max_iter=1000, C=0.1)
# aca ya estamos entrenando al modelo
model.fit(X_train, y_train)

# obtenemos las predicciones de esa prueba
y_pred = model.predict(X_test)
```

En este bloque entrenamos el modelo de Regresión Logística utilizando el conjunto de entrenamiento. Ajustamos el parámetro *C=0.1* para evitar sobreajuste y *max_iter=1000* para asegurar que el modelo converja correctamente. Una vez entrenado, el modelo se utiliza para predecir si los correos del conjunto de prueba son spam o legítimos. Estas predicciones son las que luego se comparan con las etiquetas reales para evaluar el rendimiento del modelo.

4.9 Iteraciones del modelo

```
print("Iteraciones utilizadas:", model.n_iter_)
```

```
Iteraciones utilizadas: [17]
```

El modelo utilizó 17 iteraciones durante el proceso de entrenamiento, lo que indica que la Regresión Logística logró converger de manera rápida y eficiente. Cada iteración corresponde a un ciclo en el cual el algoritmo ajusta sus parámetros para minimizar el error y mejorar la clasificación. El hecho de que alcanzara una solución óptima en tan pocas iteraciones demuestra que los datos estaban bien preparados y que el modelo no necesitó un proceso extenso para estabilizarse. En resumen, estas 17 iteraciones reflejan un entrenamiento estable, eficiente y adecuado para la complejidad del problema.

UNIVERSIDAD BERNARDO O'HIGGINS

5. Métricas y su análisis

5.1 Métricas de rendimiento

```
# Cálculamos las metricas
#Accuracy: porcentaje total de aciertos
#Precision: proporción de predicciones positivas correctas
#Recall: cuántos spam reales detecta
#F1-score: equilibrio entre precision y recall

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy :", acc)
print("Precision:", prec)
print("Recall   :", rec)
print("F1-score :", f1)

print("\nREPORTE COMPLETO:\n")
print(classification_report(y_test, y_pred))
```

En este bloque evaluamos el rendimiento del modelo utilizando métricas clave de clasificación. La *accuracy* nos muestra el porcentaje total de aciertos, mientras que la *precisión* indica qué tan correctas son las predicciones de spam. El *recall* mide cuántos correos spam reales logra detectar el modelo, y el *F1-score* resume precision y recall en una sola medida equilibrada. Finalmente, el classification_report nos entrega un resumen completo de estas métricas por clase, lo que permite analizar con mayor detalle el desempeño del modelo en correos legítimos y spam.

5.2 Matriz de confusión

```
#Aca basicamente lo que hacemos es ver:

#Verdaderos positivos (spam bien detectado)
#Verdaderos negativos (legítimos bien clasificados)
#Falsos positivos
#Falsos negativos

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicción")
plt.ylabel("Real")
plt.title("Matriz de Confusión")
plt.show()
```

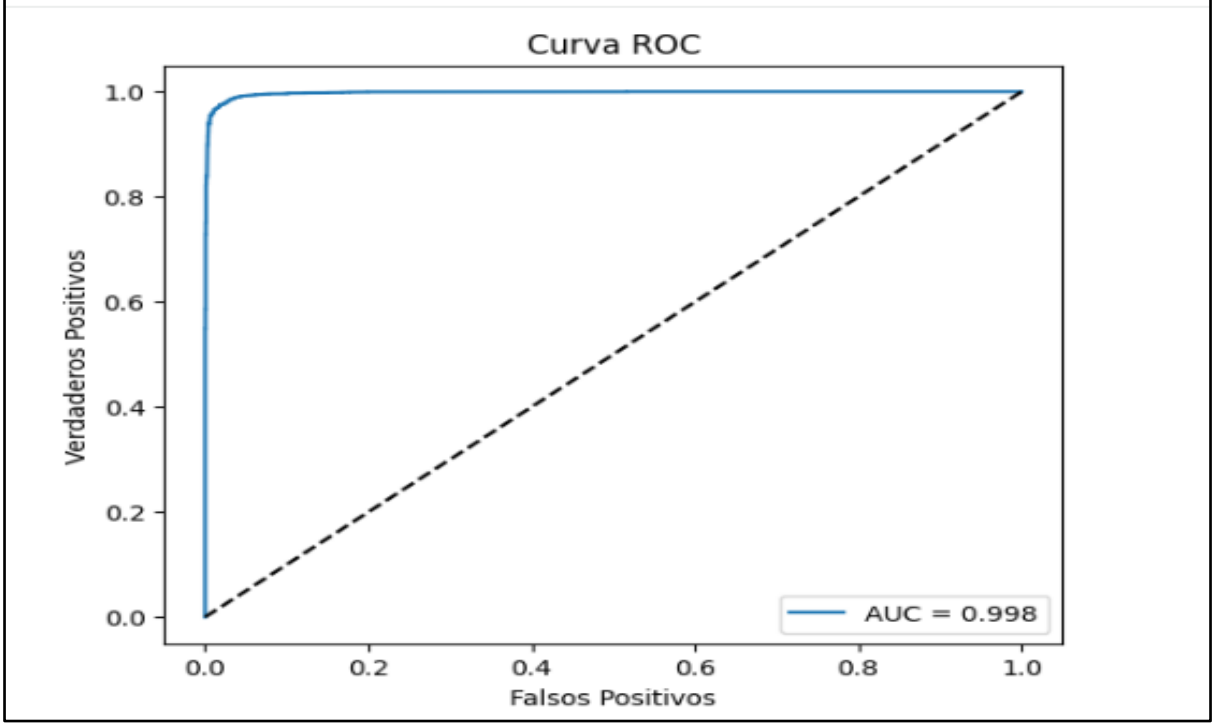
En este bloque generamos la matriz de confusión, una herramienta que permite visualizar de forma clara cómo se comportó el modelo al clasificar los correos. La matriz muestra cuántos spam fueron detectados correctamente (verdaderos positivos), cuántos correos legítimos fueron clasificados bien (verdaderos negativos) y también los errores cometidos, como falsos positivos o falsos negativos. El mapa de calor facilita interpretar estos valores, permitiendo observar de manera rápida si el modelo está confundiendo una clase con la otra.

5.3 Curva ROC

En este bloque calculamos y graficamos la curva ROC, que muestra cómo varía el rendimiento del modelo al cambiar el umbral de clasificación. Para ello, obtenemos las probabilidades de que cada correo sea spam y luego calculamos las tasas de verdaderos positivos y falsos positivos. La curva resultante permite visualizar qué tan bien separa el modelo ambas clases, y el valor AUC resume ese desempeño en un solo número. En este caso, un AUC de 0.998 indica un rendimiento excelente, demostrando que el modelo distingue muy claramente entre correos legítimos y spam.

```
# Curva ROC
y_prob = model.predict_proba(X_test)[:,1]
#ROC nos muestra cómo cambia el rendimiento según el umbral
fpr, tpr, th = roc_curve(y_test, y_prob)
# AUC nos mide el área bajo la curva
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
plt.plot([0,1], [0,1], "k--")
plt.xlabel("Falsos Positivos")
plt.ylabel("Verdaderos Positivos")
plt.title("Curva ROC")
plt.legend()
plt.show()
```



5.4 Análisis

Las métricas obtenidas muestran que el modelo tuvo un rendimiento sobresaliente en la clasificación de correos spam y legítimos. El **accuracy del 98%** indica que casi todos los correos fueron clasificados correctamente, lo que refleja un desempeño general muy alto. Sin embargo, métricas más específicas como **precisión** y **recall** entregan una visión más profunda sobre el comportamiento del modelo.

La **precisión** cercana a 0.98 señala que, cuando el modelo predice spam, casi siempre acierta, lo cual es importante para evitar falsos positivos, ya que clasificar un correo legítimo como spam puede generar pérdidas de información. Por otro lado, el **recall** también de alrededor de 0.98 muestra que el modelo logra identificar prácticamente todos los correos spam reales, reduciendo la posibilidad de que mensajes maliciosos pasen desapercibidos.

El **F1-score** combina ambas métricas y confirma este equilibrio, demostrando que el modelo no solo detecta el spam con alta eficacia, sino que lo hace sin comprometer la calidad de las predicciones. Finalmente, el valor **AUC = 0.998** evidencia una separación casi perfecta entre las dos clases, lo que significa que, independientemente del umbral utilizado, el modelo mantiene un rendimiento muy estable y confiable. En conjunto, estas métricas muestran que el modelo es altamente efectivo, preciso y adecuado para un sistema de filtrado de spam.

6. Conclusión

El desarrollo de este proyecto permitió aplicar de manera integral los conceptos de aprendizaje supervisado y procesamiento de texto para abordar el problema de clasificación de correos electrónicos spam. A través de un preprocesamiento adecuado, la vectorización con TF-IDF y el uso de un modelo de Regresión Logística, se logró construir un sistema capaz de distinguir de manera muy efectiva entre correos legítimos y maliciosos. Las métricas obtenidas —incluyendo un accuracy del 98% y un AUC de 0.998 demuestran que el modelo no solo es preciso, sino también estable y confiable en su rendimiento.

Los resultados evidencian que la combinación entre limpieza del texto, reducción de ruido y un algoritmo de clasificación bien ajustado permite alcanzar un desempeño sobresaliente en este tipo de tareas. En conjunto, se cumplió con los objetivos propuestos al inicio del trabajo y se validó que es posible implementar soluciones automatizadas para mejorar la detección de spam. Este proyecto sienta una base sólida para trabajos futuros, donde podrían explorarse modelos más complejos como redes neuronales o embeddings semánticos para seguir aumentando la precisión y robustez del sistema.