UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE TELECOMUNICACIONES

INFORME FINAL DEL TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"EVALUACIÓN DE RENDIMIENTO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO EN LA DETECCIÓN DE SPAM"

Trabajo de titulación previo a la obtención del título de Ingeniero en Telecomunicaciones

Línea de investigación: Desarrollo, aplicación de software y cybersecurity (seguridad cibernética)

AUTOR:

Darwin Oswaldo Almachi Suriaga

DIRECTOR:

Ing. Fabián Geovanny Cuzme Rodríguez MSc

Ibarra, 2025

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional, determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004748719		
APELLIDOS Y NOMBRES:	ALMACHI SURIAGA DARWIN OSWALDO		
DIRECCIÓN:	IBARRA Y MANTA		
EMAIL:	doalmachis@utn.edu.ec / darwinalmachi@gmail.com		
TELÉFONO FIJO:	XXXXXXX	**TELF. MOVIL**	0960177039

DATOS DE LA OBRA	
TÍTULO:	EVALUACIÓN DE RENDIMIENTO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO EN LA DETECCIÓN DE SPAM
AUTOR (ES):	ALMACHI SURIAGA DARWIN OSWALDO
FECHA: AAAAMMDD	21/02/2025
SOLO PARA TRABAJOS DE INTEGRACIÓN CURRICULAR	
CARRERA/PROGRAMA:	■ **GRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	INGENIERO EN TELECOMUNICACIONES
DIRECTOR:	ING. CUZME RODRÍGUEZ FABIÁN GEOVANNY, MSC
ASESOR:	ING. SUÁREZ ZAMBRANO LUIS EDILBERTO, MSC

AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, DARWIN OSWALDO ALMACHI SURIAGA, con cédula de identidad Nro. 1004748719, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de integración curricular descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 144.

Ibarra, a los 21 días del mes de febrero de 2025.

EL AUTOR:

.....

Darwin Oswaldo Almachi Suriaga

CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 21 días del mes de febrero de 2025.

EL AUTOR:

.....

Darwin Oswaldo Almachi Suriaga

CERTIFICACIÓN DEL DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 21 de febrero de 2025

ING. CUZME RODRÍGUEZ FABIÁN GEOVANNY, MSC

DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de Integración Curricular, el mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

ING. CUZME RODRÍGUEZFABIÁN GEOVANNY, MSC

C.I.: 1311527012

APROBACIÓN DEL COMITÉ CALIFICADOR

El Comité Calificador del trabajo de Integración Curricular "EVALUACIÓN DE RENDIMIENTO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO EN LA DETECCIÓN DE SPAM." elaborado por ALMACHI SURIAGA DARWIN OSWALDO, previo a la obtención del título de INGENIERO EN TELECOMUNICACIONES, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

Ing. Cuzme Rodríguez Fabián Geovanny, MSc

C.C.: 1311527012

Ing. Suárez Zambrano Luis Edilberto, MSc

C.C.: 1002304291

DEDICATORIA

Dedico esta tesis a mi padre, Pepe Almachi, quien, aunque ya no esté físicamente conmigo, sigue siendo mi mayor inspiración y guía. Su amor, enseñanzas y valores han sido la base sobre la que he construido cada logro en mi vida. Desde el cielo, sé que me acompaña en cada paso, dándome fuerzas para seguir adelante. A él, con infinito amor y gratitud, le dedico este esfuerzo, con la esperanza de honrar su memoria y todo lo que me enseñó.

A mi madre, Elvia Suriaga, por su amor incondicional y su apoyo constante, que me han dado la fuerza para superar cada obstáculo. A mis hermanos, quienes siempre han estado a mi lado, brindándome su motivación y alegría en cada momento. A todos ellos, con todo mi cariño, les dedico este logro, con la certeza de que sin su apoyo no habría sido posible.

Almachi Suriaga Darwin Oswaldo

AGRADECIMIENTO

Quisiera expresar mi más sincero agradecimiento a todos aquellos que han sido parte fundamental en la realización de esta tesis y en el desarrollo de mi carrera universitaria. A mis compañeros de estudio, quienes compartieron sus perspectivas y discutieron ideas que enriquecieron el desarrollo de este proyecto. Su colaboración y camaradería hicieron de este proceso una experiencia más llevadera y gratificante. Cada conversación, cada consejo, contribuyó significativamente al éxito de este trabajo.

A todas las personas que, directa o indirectamente, han brindado su apoyo durante este recorrido. Su tiempo, sus palabras de aliento y su ayuda han sido fundamentales. No tengo palabras suficientes para expresar mi gratitud por cada gesto que me permitió seguir adelante con determinación, tanto en mi carrera como en la culminación de esta tesis.

Agradezco enormemente a mi familia, especialmente a mi madre, Elvia Suriaga, y a mis hermanos, por su amor incondicional, su apoyo constante y su presencia en cada momento de mi vida académica. Han sido mi fuerza y motivación, y sin ellos, este logro no habría sido posible.

Mi agradecimiento más profundo al MSc Fabián Cuzme, Director de la Tesis, por su constante apoyo, paciencia y compromiso. Su orientación experta y su dedicación a lo largo de todo el proceso fueron esenciales para superar los desafíos que surgieron. También agradezco enormemente al MSc Luis Suárez, Asesor de la Tesis, por su valiosa colaboración y por ofrecerme una visión clara y precisa, lo que permitió darle un enfoque sólido a la investigación. Su disposición y profesionalismo fueron claves en el éxito de este trabajo.

RESUMEN

El presente trabajo de tesis se centró en la evaluación del rendimiento de algoritmos de aprendizaje automático para la detección de correos electrónicos de spam, un problema de gran relevancia en el ámbito de la seguridad informática y la gestión de comunicaciones digitales. Para ello, se seleccionaron tres algoritmos ampliamente utilizados en este campo: Naive Bayes, K-Nearest-Neighbor (KNN) y Decision Tree.

La investigación inició con la recopilación de un conjunto de datos representativo, asegurando una diversidad de ejemplos que reflejaran casos reales de correos clasificados como spam y no spam. Posteriormente, se llevó a cabo la implementación de los algoritmos, incluyendo su configuración inicial y ajustes específicos para maximizar su rendimiento. Como parte de este proceso, se optimizaron los parámetros de cada modelo con el fin de garantizar que operaran bajo condiciones ideales y se adaptaran a las características del conjunto de datos empleado.

El análisis comparativo de los algoritmos se realizó mediante métricas estándar de evaluación, como precisión, recall, F1-score y el área bajo la curva ROC (AUC). Naive Bayes obtuvo una precisión del 97.17%, un recall de 0.686 y un F1-Score de 0.804, mostrando un buen equilibrio. KNN alcanzó una precisión del 100%, pero con un recall de solo 0.293 y un F1-Score de 0.453, lo que limitó su efectividad en la detección de spam. Decision Tree logró una precisión del 87.91%, un recall de 0.64 y un F1-Score de 0.74, ofreciendo un rendimiento intermedio. Estos resultados permitieron identificar fortalezas y debilidades de cada modelo y formular recomendaciones para mejorar los sistemas de filtrado de correos.

Palabras clave: algoritmos de aprendizaje automático, detección de spam, Naive Bayes, K-Nearest-Neighbor, Decision Tree, filtrado de correos electrónicos.

ABSTRACT

The present thesis focused on evaluating the performance of machine learning algorithms for spam email detection, a highly relevant issue in the field of cybersecurity and digital communication management. To achieve this, three widely used algorithms in this domain were selected: Naive Bayes, K-Nearest Neighbor (KNN), and Decision Tree.

The research began with the collection of a representative dataset, ensuring a diverse range of examples that reflected real-world cases of emails classified as spam and non-spam. Subsequently, the algorithms were implemented, including their initial configuration and specific adjustments to maximize their performance. As part of this process, the parameters of each model were optimized to ensure they operated under ideal conditions and adapted to the characteristics of the dataset used.

The comparative analysis of the algorithms was conducted using standard evaluation metrics such as precision, recall, F1-score, and the area under the ROC curve (AUC). Naive Bayes achieved a precision of 97.17%, a recall of 0.686, and an F1-score of 0.804, demonstrating a good balance. KNN attained a precision of 100% but with a recall of only 0.293 and an F1-score of 0.453, limiting its effectiveness in spam detection. Decision Tree achieved a precision of 87.91%, a recall of 0.64, and an F1-score of 0.74, offering intermediate performance. These results allowed for the identification of each model's strengths and weaknesses and the formulation of recommendations to improve email filtering systems.

Keywords: machine learning algorithms, spam detection, Naive Bayes, K-Nearest Neighbor, Decision Tree, email filtering.

LISTA DE SIGLAS

ML: Machine Learning (Aprendizaje Automático)

NB: Naive Bayes

KNN: K-Nearest-Neighbor (K-Vecinos Más Cercanos)

DT: Decision Tree (Árbol de Decisión)

SPAM: Correo No Deseado

FP: False Positive (Falso Positivo)

FN: False Negative (Falso Negativo)

TP: True Positive (Verdadero Positivo)

TN: True Negative (Verdadero Negativo)

ROC: Receiver Operating Characteristic (Característica Operativa del Receptor)

AUC: Area Under the Curve (Área Bajo la Curva)

F1-Score: Métrica de Evaluación que combina Precisión y Recall

TF-IDF: Term Frequency-Inverse Document Frequency (Frecuencia de Término-Frecuencia Inversa de Documento)

NLTK: Natural Language Toolkit

UCI: Universidad de California, Irvine

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO I: ANTECEDENTES

1.1. Tema

Evaluación de rendimiento de algoritmos de aprendizaje automático en la detección de spam.

1.2. Planteamiento del problema

El correo y las comunicaciones electrónicas son componentes fundamentales de la vida cotidiana y profesional en la era digital. Sin embargo, la proliferación de mensajes no deseados o spam ha llevado a una sobrecarga de información y a la pérdida de tiempo y recursos para los usuarios. De acuerdo Muzammil et al. (2013) en los últimos años, ha habido un notable incremento en la cantidad de spam.

El spam es uno de los principales problemas de Internet y ha aumentado significativamente en los últimos años. Actualmente, más del 85% de los correos electrónicos o mensajes recibidos por los usuarios son spam (Gómez, 2021). Debido al gran volumen de datos, el análisis manual de estos mensajes no es viable. Para lograr una clasificación precisa del spam, se emplean métodos de aprendizaje automático. El correo no deseado, también conocido como spam, consiste en mensajes no solicitados enviados masivamente por spammers a través del correo electrónico (Géron, 2019).

Los principales tipos de correos electrónicos no solicitados: Publicidad; Spam nigeriano: spam utilizado por estafadores para extorsionar al destinatario de una carta; El phishing es un acto que intenta obtener electrónicamente información delicada o confidencial de los usuarios (generalmente con fines de robo) mediante la creación de un sitio web réplica de una organización legítima. Este mensaje no contiene una dirección de remitente real (Barrera, 2017).

Los dos últimos tipos de correos son los más peligrosos, ya que pueden servir como formas de propagar virus o malware. El spam puede ser utilizado como una herramienta para obtener privilegios de usuario en el sistema, con la consiguiente infiltración en este sistema y la realización de un ataque. La principal forma de influir en el usuario es la manipulación. La atención del usuario se centra en la ejecución inmediata de las acciones. Los principales métodos de atracción: dinero prometido, pagos o incluso intereses de las personas (Muzammil et al., 2013).

En la actualidad, el spam no solo es molesto, sino que también puede representar un riesgo significativo, ya que a menudo se utiliza para difundir malware, phishing y otras amenazas cibernéticas. A medida que los spammers desarrollan estrategias más sofisticadas, la detección y el filtrado de spam se convierten en un desafío cada vez más complejo.

La detección y filtrado de spam se ha convertido en un aspecto fundamental de la ciberseguridad y la gestión de correo electrónico. Según Barrera (2017) diversos métodos se han desarrollado a lo largo de los años, y numerosos algoritmos de aprendizaje automático se han aplicado para abordar este problema. Sin embargo, no todos los algoritmos funcionan de la misma manera y algunos pueden ser más efectivos que otros en la detección de spam.

Numerosas plataformas de correo electrónico, como Gmail y Outlook, emplean un algoritmo de identificación de correos no deseados para redirigirlos a la bandeja de spam. A pesar de esto, los delincuentes desarrollan estrategias para evadir la detección de spam al estudiar el funcionamiento de estos algoritmos convencionales (Géron, 2019).

Los algoritmos actuales pueden variar en términos de precisión, velocidad y capacidad para adaptarse a las cambiantes estrategias de los remitentes de spam. Además, el rendimiento de un algoritmo de detección de spam puede depender de factores como la calidad de los datos de entrenamiento y la elección de características relevantes. La evaluación y comparación de estos

algoritmos son esenciales para determinar cuáles son los más adecuados para la detección de spam en diferentes contextos (Quizhpilema, 2023).

Al abordar este problema, se busca optimizar la precisión de la detección de spam, minimizar las falsas alarmas y mejorar la eficiencia en el procesamiento de correos electrónicos, lo que en última instancia beneficiará a los usuarios y las organizaciones al garantizar una experiencia de correo electrónico más segura y eficaz.

1.3. Objetivos

1.3.1. Objetivo General

Evaluar el rendimiento de algoritmos de aprendizaje automático en la detección de spam para optimizar la calidad de la detección de correos electrónicos no deseados en un entorno de comunicación en línea.

1.3.2. Objetivos Específicos

- Analizar los fundamentos teóricos relacionados con la detección de spam mediante una revisión de la literatura, evaluando su pertinencia en el campo, con el fin establecer una base sólida para la implementación y evaluación posterior de los algoritmos seleccionados.
- Adquirir un conjunto de datos público representativo de correos electrónicos etiquetados como spam y no spam, adecuado para el entrenamiento y evaluación de los algoritmos seleccionados.
- Implementar los algoritmos seleccionados Naive Bayes, K-Nearest-Neighbor y Decision Tree, ajustar sus hiperparámetros para optimizar su rendimiento en la detección de spam.
- Comparar los resultados de los tres algoritmos para determinar cuál ofrece el mejor rendimiento en términos de precisión, sensibilidad y eficiencia en la detección de spam.

1.4. Alcance

El proyecto tiene como objetivo principal llevar a cabo la evaluación de tres algoritmos de aprendizaje automático (Naive Bayes, K-Nearest-Neighbor y Decision Tree) ampliamente utilizados en la detección de spam.

La metodología seguirá un enfoque de desarrollo de proyectos en cascada, dividiendo el trabajo en fases secuenciales: fase de investigación y análisis, adquisición de datos, implementación y ajuste de algoritmos y la fase de evaluación y comparación. La metodología seguirá un enfoque cuantitativo, se empleará un análisis de datos basado en medidas numéricas y estadísticas para evaluar el rendimiento de los algoritmos de aprendizaje automático en la detección de spam. Este método implica la recolección de datos cuantificables, como métricas de exactitud, precisión, recall y F1 Score.

La precisión, fundamental en este contexto, indicará la proporción de clasificaciones correctas. El indicador de recall evaluará la capacidad del modelo para identificar todas las instancias relevantes, crucial para minimizar falsos negativos. Para equilibrar precisión y recall, se empleará el F1 Score, ofreciendo una evaluación ponderada en situaciones de desequilibrio entre clases o cuando los falsos positivos y negativos tienen impacto significativo.

La investigación cuantitativa se centrará en la recopilación y análisis de datos numéricos provenientes de la ejecución de los algoritmos en conjuntos de prueba específicos. Se utilizarán técnicas estadísticas para comparar y contrastar los resultados, proporcionando una evaluación objetiva y cuantificable del desempeño de cada algoritmo. Este enfoque permite generalizar hallazgos a poblaciones más amplias y respaldar conclusiones basadas en evidencia numérica.

Fase de Investigación (Análisis)

En esta fase inicial, se llevará a cabo una revisión de la literatura académica y técnica relacionada con los algoritmos de detección de spam en correos electrónicos. Se identificarán los algoritmos más relevantes y se establecerán los conocimientos fundamentales para el desarrollo del proyecto. Además, se analizarán las tendencias actuales en la detección de spam y se recopilarán datos sobre estrategias de spam utilizadas en la actualidad.

Fase de Adquisición de Datos

En esta etapa, se procederá a la adquisición de un conjunto de datos público y representativo de correos electrónicos etiquetados como spam y no spam. Este conjunto de datos se utilizará para el entrenamiento y evaluación de los algoritmos seleccionados. Se garantizará la calidad y diversidad de los datos para una evaluación precisa.

Fase de Implementación y Ajuste de Algoritmos

Una vez adquiridos los datos, se implementarán los tres algoritmos de detección de spam seleccionados. Se ajustarán los hiperparámetros de cada algoritmo para optimizar su rendimiento en la detección de spam. Se trabajará en un entorno de desarrollo controlado para asegurarse de que los algoritmos funcionen correctamente.

El conjunto de datos estará dividido en tres bloques distintos: entrenamiento, validación y prueba. En la fase de entrenamiento, el modelo de detección de spam se nutre de un segmento significativo del conjunto de datos, utilizando esta información para aprender patrones y características distintivas entre correos electrónicos legítimos y spam. Este proceso busca optimizar la capacidad del algoritmo para generalizar y realizar predicciones precisas.

En la etapa de validación, se emplea otro bloque del conjunto de datos para ajustar los hiperparámetros del modelo y validar su rendimiento. Esta fase actúa como un mecanismo de ajuste fino, permitiendo la optimización del modelo antes de enfrentarlo al conjunto de prueba.

El bloque de prueba, finalmente, representa una porción independiente del conjunto de datos que el modelo no ha visto durante el entrenamiento ni la validación. La evaluación en este conjunto proporciona una medida objetiva y sin sesgos sobre la capacidad del modelo para generalizar a datos no vistos previamente, ofreciendo una evaluación realista de su rendimiento en situaciones del mundo real.

Fase de Evaluación y Comparación

En esta etapa, se realizará una evaluación de los algoritmos en un entorno lo más cercano posible a la realidad. Se utilizará un conjunto de datos de prueba que refleje situaciones reales. Se considerarán factores como la adaptabilidad a nuevas estrategias de spam y la eficiencia en la detección. Los resultados se compararán para determinar cuál algoritmo ofrece el mejor rendimiento en términos de precisión, sensibilidad y eficiencia en la detección de spam.

1.5. Justificación

La investigación de Sheth et al. (2022) destaca la importancia del correo electrónico como una herramienta esencial en la comunicación personal y empresarial en la actualidad. Sin embargo, señalan que la proliferación del spam en los correos electrónicos ha generado una sobrecarga de información y ha tenido un impacto significativo en la experiencia de los usuarios. Además, subrayan que el spam no solo resulta molesto, sino que también plantea un riesgo importante, ya que con frecuencia se utiliza como medio para difundir malware, phishing y otras amenazas cibernéticas.

De acuerdo con informes de fuentes como Muzammil et al. (2013) en los últimos años, ha habido un aumento notable en la cantidad de spam en el correo electrónico, llegando a constituir más del 85% de los correos electrónicos recibidos por los usuarios. Esta situación pone de manifiesto la necesidad urgente de abordar el problema del spam y mejorar la calidad del servicio de correo electrónico.

El proyecto se justifica en el marco de la Ley Orgánica de Protección de Datos Personales en Ecuador, que establece directrices para la salvaguardia de la privacidad en entornos digitales, especialmente en el uso extendido del correo electrónico (Asamblea Constituyente del Ecuador, 2021). Esta ley crea un contexto legal que respalda la necesidad de evaluar y mejorar algoritmos de detección de spam para garantizar la seguridad de la información personal y empresarial.

Además, el Código Orgánico Integral Penal (COIP) de Ecuador, en sus artículos 229, 230, 231, 232 y 234, aborda delitos cibernéticos, incluyendo la interceptación ilegal de datos y el acceso no consentido a sistemas informáticos (República del Ecuador, 2021). Estas disposiciones refuerzan la importancia de combatir el spam, destacando la relevancia de la investigación en la mejora de algoritmos de detección.

La investigación sobre la detección y mitigación del spam en el correo electrónico también se alinea con los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas. Específicamente, se relaciona con el ODS 16, que busca promover sociedades justas, pacíficas e inclusivas. Dentro del ODS 16, la meta 16.5 busca reducir significativamente la corrupción y el soborno en todas sus formas (Barrera, 2017). Aunque el spam en el correo electrónico no es una forma de corrupción, se puede ver como un tipo de abuso del sistema de comunicación que perjudica la integridad de la comunicación en línea. Al abordar el spam y sus amenazas asociadas,

se contribuye indirectamente a la promoción de un entorno en línea más seguro y confiable, lo que es coherente con los esfuerzos para lograr un desarrollo sostenible y justo en la era digital.

La magnitud del problema del spam se evidencia en informes, como el del Centro de Quejas de Delitos en Internet (IC3), una entidad gubernamental vinculada al FBI (Buró Federal de Investigaciones), se reportaron en los Estados Unidos un total de 241,342 víctimas de casos relacionados con el phishing y sus variantes. En ese mismo año, las pérdidas estimadas alcanzaron la suma de $54,241,075 dólares (Dada et al., 2019).

Es importante señalar que, entre los tres delitos cibernéticos más comunes en Ecuador, el phishing se destaca como el principal durante el período comprendido entre 2014 y 2020, según datos proporcionados por la Fiscalía General del Estado. Le siguen en frecuencia la falsificación o uso de documentos falsos y la apropiación fraudulenta a través de medios electrónicos (Almaguer-Perez & Hernández-Yeja, 2021).

Según Géron (2019) los spammers emplean enfoques ingeniosos y elaborados para llevar a cabo sus acciones ilegales mediante el uso de mensajes de correo no deseado. La variabilidad en la efectividad de los algoritmos y su capacidad para adaptarse a las cambiantes estrategias de los remitentes de spam destaca la necesidad de evaluar y comparar su rendimiento.

En este contexto legal y ante el incremento significativo de amenazas cibernéticas, la investigación no solo se justifica, sino que se posiciona como una contribución valiosa para mejorar la calidad del servicio de correo electrónico, proteger la privacidad de los usuarios y mitigar riesgos asociados a actividades delictivas en línea. Una detección de spam eficaz mejora la experiencia del usuario al garantizar que los correos electrónicos no deseados no interfieran con la comunicación legítima.

CAPÍTULO II. FUNDAMENTACIÓN TEÓRICA

En este capítulo, se presentaron diversos aspectos fundamentales relacionados con el correo electrónico, su estructura, protocolos asociados, así como los problemas de seguridad inherentes. Además, se profundizó en el fenómeno del spam y sus características, explorando métodos simples de detección. Posteriormente, se introdujo el concepto de Inteligencia Artificial, centrándose en ~~el~~ Machine Learning como herramienta clave en la gestión de correos electrónicos. Por último, se examinó la normativa legal pertinente.

2.1. Correo Electrónico

El correo electrónico, según la definición proporcionada por Barrera (2017), se caracteriza como un sistema de interacción mediatizada que facilita intercambios individuales o colectivos, tanto en el ámbito real como virtual. Este medio se basa en la transacción de textos digitalizados con una forma, función, estructura, lengua y estilo propios.mm En el contexto empresarial, el correo electrónico ha emergido como la herramienta de comunicación más ampliamente utilizada en la actualidad (Barrera, 2017).

Su versatilidad permite la transmisión instantánea de mensajes, imágenes y documentos a nivel global, así como el almacenamiento y archivo de estos mensajes para un mejor control de las comunicaciones. Enriqueciendo esta perspectiva, en el ámbito de las telecomunicaciones, el correo electrónico se define como un método integral para la comunicación y la transmisión de información entre dispositivos informáticos (Montero, 2020).

Es esencial resaltar que, además de su función comunicativa, el correo electrónico se clasifica como un documento de archivo debido a sus atributos de autenticidad, integridad, confiabilidad y disponibilidad. La gestión adecuada de los correos electrónicos como documentos de archivo cobra vital importancia, especialmente en la detección de spam, donde

estos correos constituyen la base de datos para evaluar el rendimiento de algoritmos de aprendizaje automático (Albán et al., 2022).

Reconocer el correo electrónico como un documento de archivo en este contexto asegura la autenticidad e integridad de los datos utilizados en la evaluación de algoritmos, destacando su relevancia más allá de su función primaria de comunicación.

2.1.1. Características del Correo Electrónico

Las características fundamentales del correo electrónico, según Barría (2023) se centran en su rapidez, ventaja económica y asincronía. La rapidez se evidencia en la entrega casi instantánea de mensajes, a diferencia del correo postal. Además, la ventaja económica radica en el bajo costo de enviar mensajes a nivel mundial. La asincronía permite que el emisor y el receptor no necesiten estar conectados simultáneamente, facilitando la redacción independiente del mensaje. Además, el correo electrónico permite la transferencia de archivos y está disponible continuamente los 365 días del año, siendo una opción ecológica al prescindir del papel en la correspondencia.

2.1.2. Cabeceras

Los encabezados de correo, según Linube (2020) constituyen información esencial en un mensaje, utilizada por los servidores de correo para identificar su origen y destino. Además, estos encabezados registran en detalle la ruta que ha seguido el mensaje, a pesar de la aparente ruta directa entre los servidores de origen y destino. Linube destaca que cada correo electrónico atraviesa al menos cuatro servidores distintos en su trayecto.

Linube aconseja, al analizar una cabecera de correo, comenzar por el final y avanzar hacia el inicio. La cabecera incluye campos como "From", "To", "Reply-Path", entre otros (Linube, 2020).

Figura 1

Formato estándar de un correo electrónico y su estructura de cabeceras

Nota. La figura representa un ejemplo típico de correo electrónico con su estructura de cabeceras. La representación gráfica destaca la disposición estándar de los elementos clave en la interfaz de correo electrónico (Linube, 2020).

Campos como "From", "To" y "Reply-Path" proporcionan información sobre el remitente, destinatario y la dirección para respuestas. Es importante tener en cuenta que la identidad del remitente puede ser falsa, permitiendo la suplantación de direcciones de correo electrónico. Linube menciona que muchos mensajes no deseados (spam) se envían con información de remitente falsificada, utilizando el campo "From" para especificar la dirección que desean suplantar y "Reply-To" para dirigir las respuestas hacia los spammers. Ejemplos comunes incluyen direcciones como "From: "REMITENTE" remitente@dominioremitente.com" y "Reply-To y Return-Path remitente@dominioremitente.com"(Linube, 2020).

Otros campos incluyen "Date" que muestra la fecha de envío, "Subject" que indica el asunto del correo, "X-Spam" que se refiere a las cabeceras adicionales de sistemas antispam,

"Received" que detalla la ruta del correo desde el origen hasta la bandeja de entrada, "by" que indica el servidor de destino, "Content-type" que especifica el formato del mensaje, y "X-Spam-Level" y "X-Spam-Status" que indican la puntuación de spam asignada por el servicio de correo o cliente utilizado.

2.1.3. Protocolos del Correo Electrónico

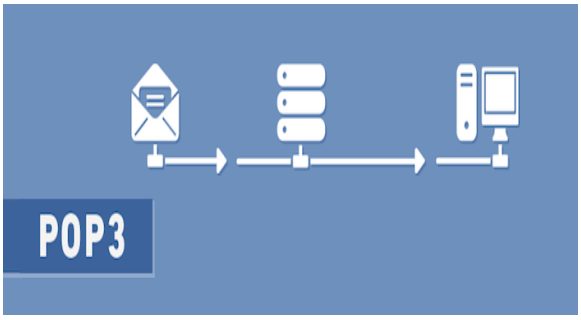
Los protocolos de correo electrónico juegan un papel crucial en la comunicación y el intercambio de información en la era digital, estableciendo normas y procedimientos para la transmisión de datos entre diversos dispositivos y software (Quizhpilema, 2023). Entre los protocolos más comunes se encuentran el Protocolo POP3 y el Protocolo IMAP.

2.1.3.1. Protocolo POP 3

El Protocolo POP 3, como se detalla en la investigación de Quizhpilema (2023) se utiliza extensamente para la lectura y descarga de correos electrónicos en el dispositivo local. Aunque ofrece la ventaja de descargar toda la información en el disco duro del cliente, lo que implica que el servidor no retiene copias de los mensajes, presenta la desventaja de limitar el acceso a los correos desde otros dispositivos después de su descarga local.

Figura 2

Proceso de comunicación mediante el protocolo POP3



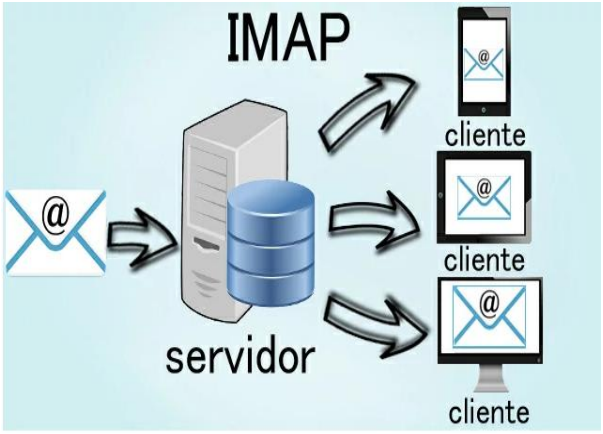
Nota. En el proceso de comunicación mediante el protocolo POP3, el cliente establece una conexión, se autentica, accede a los buzones, descarga los correos y actualiza el estado del servidor antes de cerrar la conexión. Tomado de (Mexagon, 2018).

2.1.3.2. Protocolo IMAP

El Protocolo IMAP, según la explicación de Díez (2021) permite el acceso a correos electrónicos almacenados en un servidor remoto desde cualquier dispositivo con conexión a internet. A diferencia del POP3, en este protocolo los correos no se descargan ni almacenan en el equipo local, sino que se visualizan directamente desde el servidor remoto. Esto facilita a los usuarios acceder a sus correos desde cualquier dispositivo con conexión a internet, proporcionando mayor conveniencia para aquellos que necesitan acceder a su correo electrónico desde diferentes dispositivos.

Figura 3

Intercambio de información a través del protocolo IMAP



Nota. El protocolo IMAP facilita la comunicación entre un cliente de correo y un servidor, permitiendo la gestión flexible de mensajes directamente en el servidor y sincronización de estado entre dispositivos. Tomado de (Montero, 2020).

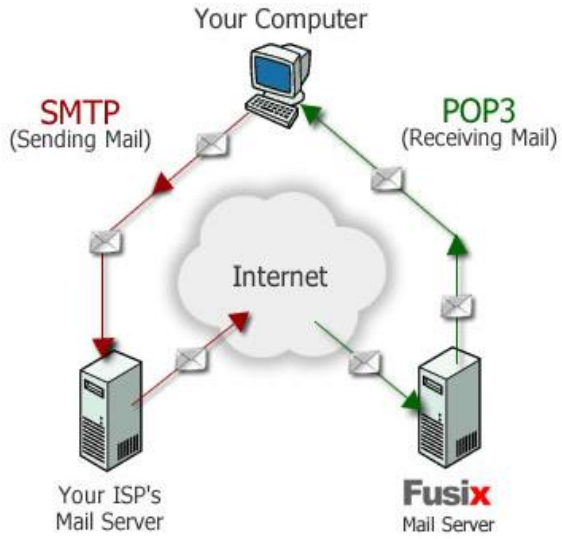
2.1.3.3. Protocolo SMTP

El Protocolo SMTP, según la documentación de (Riabov, 2017) es esencial para el envío y recepción de correos electrónicos. Este protocolo se emplea junto con POP3 o IMAP para almacenar mensajes en un servidor de correo electrónico y luego descargarlos según sea necesario. La función principal del Protocolo SMTP radica en facilitar la comunicación de correo

electrónico, permitiendo la accesibilidad de la información mediante el envío y la recepción de mensajes electrónicos.

Figura 4

Proceso de comunicación mediante el protocolo SMTP



Nota. SMTP se utiliza para enviar correos electrónicos desde un cliente de correo electrónico a un servidor de correo saliente. Es responsable de la transferencia y enrutamiento de mensajes salientes hacia el servidor de destino. Tomado de (Dipak, 2018).

2.1.4. Problemas de Seguridad en los Correos Electrónicos

La garantía de un servicio de correo electrónico completamente seguro resulta utópica, ya que estos servicios, ya sea en entornos de intranet o internet, son susceptibles a ataques perpetrados por individuos malintencionados. Estos ataques no solo pueden ocasionar perjuicios a los usuarios, sino también a la propia institución que proporciona el servicio, comprometiendo la privacidad y la integridad de la información transmitida a través de este medio (Almaguer-Perez & Hernández-Yeja, 2021).

Los servicios de correo electrónico, al igual que cualquier otro servicio en línea, se encuentran expuestos a diversas amenazas de seguridad que pueden derivar en consecuencias

perjudiciales. La vulnerabilidad ante ciberataques implica riesgos potenciales, como la interceptación de mensajes confidenciales, la manipulación de datos, el robo de información sensible o la introducción de malware en los sistemas.

Entre las principales amenazas que acechan a los usuarios e instituciones a través del servicio de correo electrónico, se incluyen la ingeniería social, el phishing, la distribución de archivos maliciosos adjuntos, el robo de credenciales y la suplantación de identidad. Estas tácticas maliciosas pueden poner en peligro tanto la seguridad de la información como la reputación de la entidad afectada (Almaguer-Perez & Hernández-Yeja, 2021).

Por ende, la implementación de medidas de seguridad robustas, como la autenticación multifactor, la encriptación de extremo a extremo y la concientización constante de los usuarios acerca de las prácticas seguras en el uso del correo electrónico, se vuelve imperativa para mitigar los riesgos asociados a este servicio esencial.

2.2. Spam

El spam, según la definición proporcionada por Dada et al. (2019) se refiere al envío masivo de mensajes a través de Internet. Este tipo de mensajes se considera una molestia significativa y representa un desafío considerable para los proveedores de servicios de Internet (ISP) en términos de filtrado y bloqueo, generando inconvenientes y conflictos en el entorno digital. El spam abarca correos electrónicos no deseados, así como mensajes en redes sociales y otras plataformas en línea.

El impacto del spam se traduce en la pérdida de tiempo para los usuarios, el uso innecesario de recursos informáticos y la facilitación de la propagación de programas maliciosos. La regulación legal del spam se presenta como un aspecto fundamental para salvaguardar los derechos de los ciudadanos y abordar los diversos perjuicios que este fenómeno ocasiona.

En la actualidad, la presencia de contenidos de spam en los medios sociales está en aumento, lo que resalta la importancia crucial de la detección y el control del spam. Para abordar este desafío, se están empleando diversas técnicas de aprendizaje automático y aprendizaje profundo en la detección y filtración de spam en diversas plataformas en línea (Dada et al., 2019).

2.2.1. Características del Spam

Según Bhowmick & Hazarika (2018) las características distintivas del spam abarcan una serie de aspectos que generalmente incluyen el envío de correos no solicitados, dirigidos a destinatarios sin su previo consentimiento. Este tipo de mensajes indeseados tiende a contener contenido no deseado, como publicidad no solicitada, promociones engañosas o información irrelevante. Además, muchos correos de spam buscan engañar o estafar a los destinatarios mediante prácticas como el phishing, donde se hacen pasar por entidades legítimas con el objetivo de obtener información personal o financiera.

La naturaleza masiva del spam se manifiesta en su envío a grandes cantidades de destinatarios, con el propósito de llegar al mayor número posible de personas. Estos mensajes a menudo incluyen enlaces sospechosos o URLs falsas que pueden redirigir a usuarios a sitios web maliciosos. La presencia de gramática y ortografía deficientes en estos correos puede servir como indicador de su ilegitimidad. Además, el uso de remitentes desconocidos o falsos con direcciones de correo electrónico ficticias es común en el spam, contribuyendo a la ocultación de la identidad del remitente. En algunos casos, la dificultad para darse de baja de las listas de correo de spam complica que los destinatarios dejen de recibir estos mensajes indeseados (Holt, 2019).

2.2.2. Métodos de Filtrado de Spam

Los siguientes métodos de filtrado de spam son los más populares hoy en día (Bhowmick & Hazarika, 2018):

- Sistemas con una solicitud de confirmación: Se solicita al remitente que realice alguna acción para garantizar la entrega del mensaje original, de lo contrario, el mensaje se considera no entregado.
- Uso de direcciones postales temporales: El usuario cambia la dirección en caso de que haya un gran número de cartas entrantes.
- Lista negra. Cuando llega un mensaje entrante, el filtro de spam comprueba si su dirección IP o de correo electrónico está en la lista negra; Si es así, el mensaje se considera spam y se rechaza.
- Lista blanca. El principio de funcionamiento es el mismo que en el método con listas negras, pero la comprobación se realiza para la ausencia de la dirección IP de envío en la lista negra del servidor de correo.
- Reconocimiento de spam basado en firmas: Una firma es una imagen o característica de un mensaje de correo electrónico. Para cada nuevo, se calcula su firma y se compara con la base de datos, que almacena las características de los mensajes previamente clasificados como spam. Si la firma del mensaje coincide con uno de los registros de la base de datos, el mensaje se considera spam.
- Heurísticas lingüísticas: Busca en el cuerpo del mensaje palabras clave y frases que permitan atribuir este mensaje a spam.

2.2.2.1. Enfoque de aprendizaje automático para el filtrado de correo electrónico no deseado

El aprendizaje automático es un subconjunto extenso de la IA que estudia cómo construir algoritmos que puedan aprender y hacer predicciones, en otras palabras, extraer patrones de ejemplos. En la siguiente sección, echaremos un vistazo a algunas de las técnicas de aprendizaje automático más populares para clasificar el spam, así como a los enfoques existentes para la detección de spam (Chipana et al., 2023).

2.2.2.2. Etapas del aprendizaje automático

Para empezar, describiremos las principales etapas del proceso de aprendizaje automático. En primer lugar, la etapa de análisis: en esta etapa, trabajamos con datos que se procesan, analizan y se revelan patrones. En segundo lugar, la etapa de entrenamiento: se utilizan modelos de aprendizaje automático sobre los datos obtenidos. La calidad de los modelos se puede mejorar mediante la selección de hiperparámetros. La siguiente etapa es la prueba: los modelos de aprendizaje automático se prueban en datos no utilizados. Se pueden utilizar varias métricas para evaluar el modelo. La última etapa es la aplicación: despliegue del mejor modelo (Uvence Rodríguez, 2020).

2.2.3. Métodos Simples de Detección de Spam

En el ámbito de la detección de correo spam, se emplean diversas estrategias, según Dada et al. (2019) que se fundamentan en estrategias o reglas prácticas y en la observación de atributos extraídos de los correos electrónicos. Se destacan dos enfoques primordiales: las aproximaciones colaborativas y los modelos basados en contenido.

2.2.3.1. Aproximaciones Colaborativas

En el contexto de la detección de aproximaciones colaborativas, se destacan dos mecanismos principales. En primer lugar, se encuentran las Listas Negras y Blancas, que se basan en reglas simples para filtrar mensajes manipulados o provenientes de dominios específicos. Las listas negras contienen información compartida por usuarios sobre mensajes spam, identificando direcciones o dominios no deseados, mientras que las listas blancas enumeran equipos considerados de confianza garantizada. En segundo lugar, se aborda el uso de Resúmenes, donde aplicaciones como Razor, Pyzor y DCC emplean algoritmos como Nilsimsa para analizar resúmenes de correos electrónicos no deseados. Estos algoritmos permiten sortear pequeñas modificaciones, facilitando la detección de spam al analizar el mensaje en su totalidad y eliminar variantes triviales (Almaguer-Perez & Hernández-Yeja, 2021).

2.2.3.2. Modelos Basados en Contenido

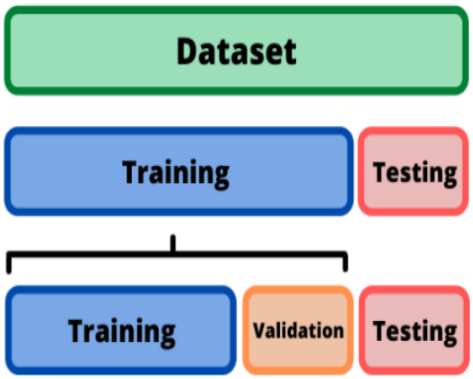
Dentro de la categoría de modelos basados en contenido, se emplean dos enfoques clave. En primer lugar, la selección de características implica la extracción de información de la cabecera o el cuerpo del mensaje para su clasificación, utilizando representaciones en forma de vector de características y seleccionando palabras representativas mediante técnicas como la ganancia de información y la información mutua. Segundo, los modelos basados en contenido incorporan la frecuencia de documentos (DF) como métrica de relevancia, calculando la frecuencia de un término en el conjunto de correos para evaluar su importancia en la detección de spam. Esta medida proporciona una evaluación cuantitativa de la representatividad de términos específicos en la clasificación de mensajes no deseados (Dada et al., 2019).

2.2.3.3. Machine Learning

El machine learning, según (Chipana et al., 2023) constituye una disciplina dentro de la Inteligencia Artificial (IA) que concede a las computadoras la habilidad de aprender sin requerir la programación explícita de las acciones a ejecutar. En este proceso, el machine learning utiliza datos para identificar patrones y ajustar las acciones del programa correspondiente. Los datos empleados pueden presentarse en diversos formatos, tales como datasets, imágenes, vídeos o audios. La metodología básica de machine learning implica la división del conjunto de datos inicial en 2 o 3 subconjuntos, conocidos como entrenamiento, validación y prueba, respectivamente.

Figura 5

Enfoque fundamental de aprendizaje automático.



Nota. Este enfoque de tres fases asegura que el modelo se desarrolle de manera robusta y generalice bien a nuevos datos, al tiempo que permite ajustar su complejidad y configuración mediante la validación. Tomado de (Pandey, 2022).

Esta división es esencial para la construcción de un modelo que pueda generalizar de manera efectiva, evitando la limitación a trabajar únicamente con los datos utilizados en su creación, Con este propósito, los dos primeros conjuntos se utilizan para diseñar el modelo,

mientras que el último se reserva para evaluar su capacidad predictiva (Bhowmick & Hazarika, 2018).

2.2.3.1. Los algoritmos

Se seleccionaron cinco algoritmos de clasificación ampliamente extendidos en el aprendizaje automático: Naive Bayes, K- Vecinos más cercanos, Máquina de vectores de soporte, Regresión logística, Árbol de decisión, Bosque aleatorio.

- **Naive Bayes**

El Teorema de Bayes se emplea para desarrollar clasificadores Bayes ingenuos, que forman un conjunto de métodos de clasificación. Estos algoritmos comparten el mismo principio fundamental: cada par de características que se va a clasificar se considera independiente. En el caso del clasificador Naive Bayes (NB), la regla de Bayes se utiliza de la siguiente manera:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)} \qquad (1)$$

Donde y es una variable de clase y, X es un vector de características dependiente n-dimensional.

$$X = (X_1, X_2, X_3, \ldots, X_n) \qquad (2)$$

En nuestro contexto, la variable de clase (y) tiene dos resultados posibles: sí o no, aunque la clasificación puede ser multivariante en algunas circunstancias. Por lo tanto, el objetivo es determinar la clase Y con la probabilidad más alta.

$$y = p(y)\prod_{t=1}^{n}\prod p(X|y) \qquad (3)$$

La precisión de los valores proyectados se utiliza para evaluar el error del procedimiento. En el caso de valores objetivo-categóricos, el error se expresa como una tasa de error, que representa la proporción de veces que la predicción fue incorrecta (Sheth et al., 2022).

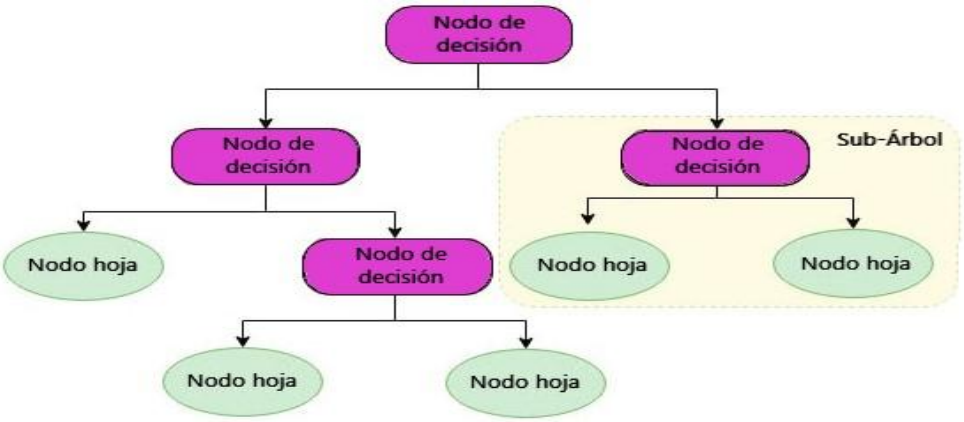
El clasificador Naive Bayes destaca por su facilidad de configuración, buen rendimiento, escalabilidad proporcional a la cantidad de predictores y puntos de datos, menor necesidad de datos de entrenamiento, capacidad para manejar datos discretos y continuos, aptitud para abordar problemas de clasificación binaria y de clases múltiples, así como la capacidad de realizar recomendaciones estocásticas. Además, puede procesar datos de forma continua o discontinua y no se ve afectado significativamente por características no esenciales. Es importante tener en cuenta que Naive Bayes asume independencia condicional, lo que implica que la relación entre todas las características de entrada es considerada como independiente (Bhowmick & Hazarika, 2018).

- **Decision Tree**

Según Sheth et al. (2022) un árbol de decisión representa una técnica de aprendizaje supervisado utilizada principalmente para tareas de clasificación y, en menor medida, para regresión. Su estructura comprende un nodo raíz, nodos de rama y nodos de hoja, configurados de manera similar a un árbol. Cada nodo representa una característica o atributo, cada rama denota una decisión o regla, y cada hoja representa un resultado. La división de características se realiza mediante algoritmos específicos para árboles de decisión, evaluando en cada nodo la idoneidad de la división para las clases respectivas.

Figura 6

Diagrama del algoritmo árbol de decisión



Nota. En este modelo, se construye una estructura en forma de árbol donde cada nodo representa una decisión basada en características específicas de los datos. Tomado de (Gómez, 2021).

El árbol de decisión se presenta como un diagrama gráfico que proporciona respuestas basadas en la situación actual, centrándose en una pregunta y dividiendo el árbol en subárboles según la respuesta. Entre los beneficios que ofrece se encuentran su eficacia tanto para problemas de regresión como de clasificación, su facilidad de interpretación, la capacidad para completar datos incompletos mediante el valor más probable en atributos y la gestión de valores categóricos y cuantitativos. Además, destaca por su productividad mejorada gracias a la eficiencia del algoritmo de recorrido del árbol.

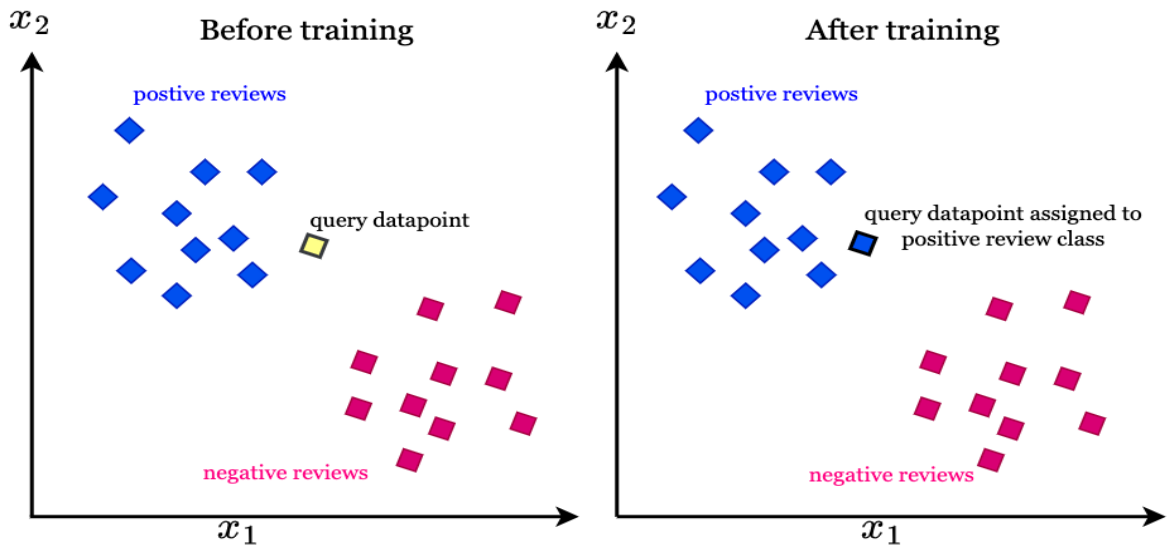
Sin embargo, el árbol de decisión enfrenta desafíos, como el sobreajuste, para el cual se propone Random Forest como solución, basado en una técnica de modelado de conjuntos. Entre las desventajas se encuentran su inestabilidad, la dificultad para gestionar el tamaño del árbol, la propensión a errores en el muestreo y la tendencia a proporcionar una respuesta óptima local en lugar de una solución globalmente ideal (Sheth et al., 2022).

- **K-Nearest-Neighbor**

En el ámbito del aprendizaje automático supervisado, los algoritmos de los K vecinos más cercanos (KNN) se destacan por su versatilidad para abordar tanto problemas de clasificación como de regresión. Estos algoritmos permiten una rápida clasificación de nuevos datos en categorías predefinidas. El modelo KNN utiliza la noción de "similitud de características" para estimar los valores de un punto de datos nuevo. Al evaluar las distancias entre una consulta y cada ejemplo en los datos, selecciona los K ejemplos más cercanos a la consulta y determina la etiqueta con la frecuencia más alta en el caso de clasificación, o promedia las etiquetas en el caso de regresión (Roshna, 2024).

Figura 7

K-Nearest Neighbors (K-NN) utilizado para problemas de clasificación de regresión



Nota. K-Nearest Neighbors (KNN) asigna una etiqueta o valor numérico a una nueva instancia basándose en las etiquetas de los K vecinos más cercanos, determinados por la medida de distancia en el espacio de características. Tomado de (Roshna, 2024)

En el proceso de aprendizaje, KNN analiza una tupla de prueba junto con tuplas de entrenamiento comparables, utilizando un espacio de patrón de n dimensiones que contiene todas

las tuplas de entrenamiento. Un clasificador KNN examina este espacio de patrón para identificar los K vecinos más cercanos a la tupla desconocida. Según Muzammil et al. (2013) el algoritmo KNN presenta ventajas como su simplicidad y rápida implementación, siendo una técnica de construcción de modelos económica y adaptable, especialmente adecuada para clases multimodales con varias etiquetas de clases en los registros. Aunque la tasa de error de KNN puede ser el doble de la tasa de error de Bayes, en algunos casos, como la predicción de la función de proteínas basada en perfiles de expresión, KNN ha demostrado superar a SVM.

Sin embargo, el algoritmo KNN no está exento de desventajas. Clasificar registros desconocidos resulta relativamente costoso, ya que implica calcular la distancia entre los K vecinos más cercanos. Además, su costo computacional aumenta con el tamaño del conjunto de entrenamiento, y la precisión puede degradarse debido a características ruidosas o irrelevantes (Sheth et al., 2022).

KNN es un algoritmo de clasificación. Está sujeto a algoritmos supervisados. Se supone que todos los puntos de datos están en un espacio n-dimensional. Y luego, en función de los vecinos, la categoría de los datos actuales se determina en función de la mayoría. La distancia euclidiana se utiliza para determinar la distancia entre puntos, como se muestra en la ecuación 4.

La distancia entre 2 puntos se calcula de la siguiente manera:

$$d = \sqrt{[(\mathrm{x2} - \mathrm{x1})]^2 + [(\mathrm{y2} - \mathrm{y1})]^2} \qquad (4)$$

Se calculan las distancias entre el punto desconocido y todos los demás. Dependiendo de la K proporcionada, se determinan los k vecinos más cercanos. La categoría a la que pertenecen la mayoría de los vecinos se selecciona como categoría de datos desconocidos.

Si los datos contienen hasta 3 entidades, se puede visualizar el gráfico. Es bastante lento en comparación con otros algoritmos basados en la distancia, como SVM, ya que necesita

determinar la distancia a todos los puntos para obtener los vecinos más cercanos al punto dado (Sheth et al., 2022).

- **Regresión logística**

La regresión logística es un algoritmo de "aprendizaje automático supervisado" que se puede utilizar para modelar la probabilidad de una determinada clase o evento. Se utiliza cuando los datos son linealmente separables y el resultado es binario o dicotómico. Las probabilidades se calculan mediante una función sigmoide.

Por ejemplo, tomemos un problema en el que los datos tienen n características. Necesitamos ajustar una línea para los datos dados, y esta línea se puede representar mediante la siguiente ecuación:

$$\mathrm{Z} = \mathrm{b}_0 + \mathrm{b}_1 + \mathrm{b}_2\mathrm{x}_2 + \mathrm{b}_3\mathrm{x}_3 + \cdots + \mathrm{b}_\mathrm{n}x_n \qquad (5)$$

En la ecuación 5, Z representa la combinación lineal de las características del modelo, $x_1, x_2, \ldots x_\mathrm{n}$, representan las características o variables independientes, y $b_0, b_1 \ldots b_\mathrm{n}$ son los coeficientes del modelo:

- b_0: Es el término de intercepto o sesgo del modelo. Representa el valor de z cuando todas las características $x_1, x_2, \ldots x_\mathrm{n}$, son cero.
- $b_1, b_2 \ldots b_\mathrm{n}$: Son los coeficientes que se multiplican por cada una de las características $x_1, x_2, \ldots x_\mathrm{n}$. Estos coeficientes indican la influencia de cada característica en la probabilidad del resultado. Un coeficiente positivo implica que un aumento en esa característica aumenta la probabilidad del evento, mientras que un coeficiente negativo implica lo contrario.

Función sigmoid: Una función sigmoide es una forma especial de función logística, de ahí el nombre de regresión logística. El logaritmo de las probabilidades se calcula y se introduce en la función sigmoide para obtener una probabilidad continua que oscila entre 0 y 1.

El logaritmo de las cuotas se puede calcular de la siguiente manera:

log (probabilidades) = punto (características, coeficientes) +intercepto *(6)*

Y estos logs_odds se utilizan en la función sigmoide para obtener probabilidades.

$$h(z) = \frac{1}{1 + e^{-z}} \qquad (7)$$

La salida de la función sigmoide es un número entero en el rango de 0 a 1 que se utiliza para determinar a qué clase pertenece la muestra. Por lo general, 0,5 se considera el límite por debajo del cual se considera un NO, y 0,5 o superior se considerará un SÍ. Pero el límite se puede ajustar en función del requisito (Sheth et al., 2022).

- **Máquinas de vectores de soporte (SVM)**

Es un algoritmo de aprendizaje automático para la clasificación. Los límites de decisión se trazan entre varias categorías y, en función del lado en el que cae el punto hasta el límite, se determina la categoría.

Vectores de soporte

Los vectores más cercanos a los límites se denominan vectores/planos de soporte. Si hay n categorías, entonces habrá n+1 vectores de soporte. En lugar de puntos, se denominan vectores porque se supone que parten del origen. La distancia entre los vectores de soporte se denomina margen (Marín et al., 2021). Queremos que nuestro margen sea lo más amplio posible porque da mejores resultados.

Hay tres tipos de límites utilizados por SVM para crear límites.

Lineal: se utiliza si los datos son linealmente separables.

Poly: se utiliza si los datos no son separables. Convierte cualquier dato en datos tridimensionales.

Radial: este es el kernel predeterminado utilizado en SVM. Convierte cualquier dato en datos de dimensión infinita.

Si los datos son bidimensionales, los límites son líneas. Si los datos son tridimensionales, los límites son planos. Si las categorías de datos son más de 3, los límites se denominan hiperplanos.

Una SVM depende principalmente de los límites de decisión de las predicciones. No compara los datos con todos los demás datos para obtener la predicción debido a que las SVM tienden a ser rápidas con las predicciones (Marín et al., 2021).

- **Enfoques y descripción del experimento**

Hay dos enfoques principales para la detección de spam:

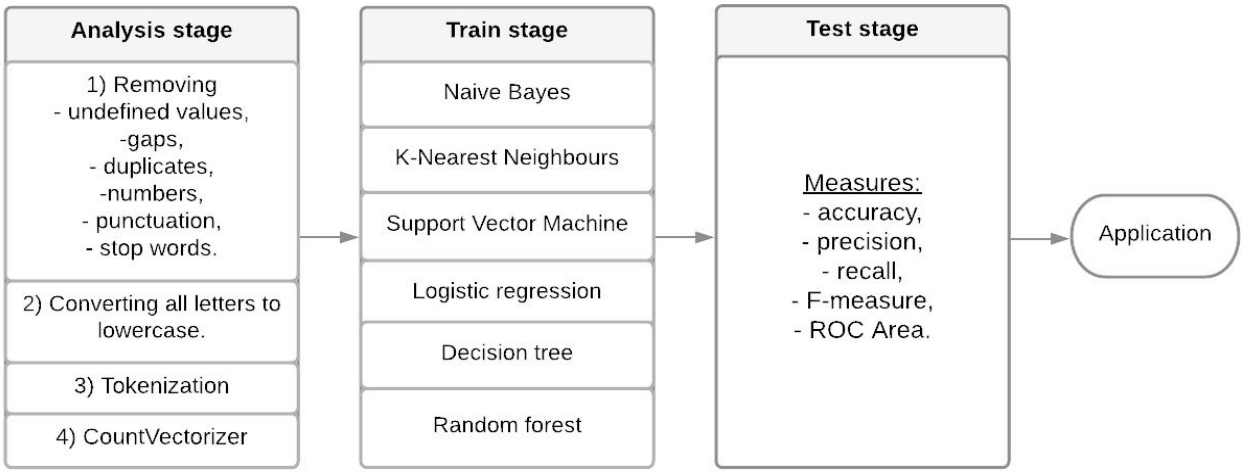
1) Clasificación de imágenes u otros archivos adjuntos mediante algoritmos de aprendizaje automático para identificar amenazas.

2) Procesamiento de lenguaje natural para analizar el texto de un correo electrónico con el fin de detectar spam.

El segundo enfoque se elige para crear el algoritmo de detección de spam. La Fig. 8 describe las principales etapas del aprendizaje automático y los subpuntos a los que se exponen los datos.

Figura 8

Esquema del algoritmo desarrollado



Nota. Análisis comparativo de algoritmos de aprendizaje automático con fines de clasificación Tomado de (Sheth et al., 2022).

En la primera etapa del aprendizaje automático, los datos se limpiarán, luego se separará una oración en palabras y se construirá un sistema de atributos para un texto. Después de eso, los algoritmos seleccionados serán entrenados y su exactitud será evaluada utilizando medidas como exactitud, precisión, recuperación, medida F y área ROC. De esta manera, se seleccionará un algoritmo que resuelva mejor el problema de la clasificación del spam (Sheth et al., 2022).

2.2.4. La Inteligencia Artificial

Según Russell & Norvig (2021) la inteligencia artificial (IA) se conceptualiza como "la concepción y construcción de agentes inteligentes capaces de recibir percepciones del entorno y llevar a cabo acciones que afecten dicho entorno". En esta definición, se destaca la capacidad de los sistemas de IA para interactuar con su entorno y tomar decisiones basadas en las percepciones que reciben.

Adicionalmente, los autores resaltan que, a diferencia de disciplinas como la Filosofía o la Psicología, el enfoque de la IA no se limita a comprender el pensamiento humano, sino que

también busca construir entidades con la capacidad de realizar dicho pensamiento. En este contexto, la IA se orienta hacia la creación de programas que puedan emular el razonamiento humano, tomar decisiones y ejecutar acciones. Esta perspectiva amplía el horizonte de la IA más allá de la simple imitación de la inteligencia humana, abriendo la posibilidad de desarrollar sistemas capaces de abordar tareas complejas de manera autónoma.

La inteligencia artificial (IA) engloba diversas áreas que convergen para desarrollar sistemas con la capacidad de emular habilidades cognitivas humanas. Entre estas disciplinas se destacan según Smarandache (2022), el aprendizaje automático, que se enfoca en entrenar algoritmos para reconocer patrones a partir de datos; el procesamiento del lenguaje natural, que facilita la interacción entre máquinas y el lenguaje humano; la visión por computadora, que posibilita la interpretación de información visual; y la robótica, que fusiona la IA con la ingeniería para crear sistemas autónomos. Además, se incluyen campos como las redes neuronales y el aprendizaje profundo, la ética de la IA, la ingeniería del conocimiento, la planificación y toma de decisiones, la optimización y los sistemas expertos. Estas áreas colaboran para afrontar diversos desafíos, desde la optimización de algoritmos hasta la consideración de implicaciones éticas y sociales en la implementación de la inteligencia artificial (Russell & Norvig, 2021).

2.2.4.1. Clasificación de Métodos de Aprendizaje

- **Aprendizaje supervisado**: Se define por trabajar con datos etiquetados, es decir, además de los datos necesarios para realizar la predicción, se requiere conocer la característica objetivo para cada instancia. Estos datos suelen ser valores históricos que permiten al modelo aprender a etiquetar los datos de manera precisa. La etiqueta se

emplea exclusivamente durante el proceso de entrenamiento del modelo, mientras que en la etapa de prueba se evalúa la calidad de la predicción (Géron, 2019).

- **Aprendizaje no supervisado:** Se caracteriza por la ausencia de datos etiquetados, ya que su objetivo es descubrir nuevos patrones o resultados a partir de los datos. Este tipo de problemas tienden a ser más complejos, ya que el modelo debe aprender sin tener conocimiento de la característica objetivo de cada instancia (Géron, 2019).
- **Aprendizaje profundo:** El Deep Learning, también conocido como Aprendizaje Profundo, es una rama del Machine Learning que ha despertado interés debido a su versatilidad. Su avance se debe al desarrollo tecnológico, especialmente a potentes GPUs que permiten un entrenamiento rápido de modelos. Se basa en investigaciones biológicas sobre neuronas en el cerebro humano, donde una red neuronal artificial se compone de neuronas interconectadas mediante enlaces (Goodfellow et al., 2016). En el ámbito del machine learning, abordar la detección de spam se conceptualiza como un problema de clasificación binaria para determinar si un correo electrónico es malicioso o no.

2.2.5. Normativa Legal

El marco legal en Ecuador proporciona un enfoque integral para la protección de datos personales, la seguridad digital y la lucha contra el spam. La Constitución de la República del Ecuador, en su artículo 66, numeral 14, consagra el derecho a la inviolabilidad de la correspondencia y comunicaciones privadas, extendiendo esta protección a los mensajes electrónicos (Asamblea Constituyente del Ecuador, 2008). Asimismo, la Ley de Comercio Electrónico, Firmas y Mensajes de Datos de Ecuador reconoce los mensajes de datos y las firmas electrónicas como medios de prueba, fortaleciendo la validez legal de la comunicación digital (República del Ecuador, 2012).

La Ley Orgánica de Protección de Datos Personales establece una serie de principios y obligaciones para el tratamiento de datos personales. Estos principios y obligaciones se aplican a cualquier tratamiento de datos personales (Asamblea Constituyente del Ecuador, 2021). En consonancia con la protección de datos, el Código Orgánico Integral Penal (COIP) de Ecuador aborda de manera específica los delitos cibernéticos en sus artículos 229, 230, 231, 232 y 234. Estos artículos se centran en la interceptación ilegal de datos, el acceso no consentido a sistemas informáticos, el ataque a la integridad de un sistema informático y la apropiación fraudulenta por medios electrónicos (República del Ecuador, 2021).

En el ámbito de las telecomunicaciones, la Ley Orgánica de Telecomunicaciones (LOT) de Ecuador, en su artículo 108, establece que los proveedores de servicios de telecomunicaciones tienen la obligación de implementar mecanismos para prevenir y bloquear el envío de spam. Estos mecanismos deben ser efectivos y deben respetar los derechos de los usuarios (República del Ecuador, 2016). La LOT establece obligaciones clave para los operadores, incluyendo la obligación de calidad del servicio, transparencia y seguridad, especialmente en lo que respecta a la detección de spam.

Estas disposiciones refuerzan la necesidad de combatir el spam y subrayan la relevancia de la investigación para mejorar los algoritmos de detección, al tiempo que establecen un marco legal robusto para combatir los delitos informáticos.

CAPÍTULO III. METODOLOGIA E IMPLEMENTACIÓN

En este capítulo, se delineó la metodología y el enfoque para evaluar los algoritmos de detección de spam. Se comenzó con la selección de algoritmos adecuados y la preparación detallada de los conjuntos de datos de entrenamiento y prueba, estableciendo así una base sólida para la implementación efectiva de los modelos. Se dedicó especial atención al proceso de implementación y ajuste de los algoritmos, asegurando su optimización para una detección precisa de spam. La inclusión de técnicas de validación cruzada y medidas de rendimiento rigurosas garantizó una evaluación objetiva y completa del desempeño de los modelos.

3.1. Metodología del proyecto

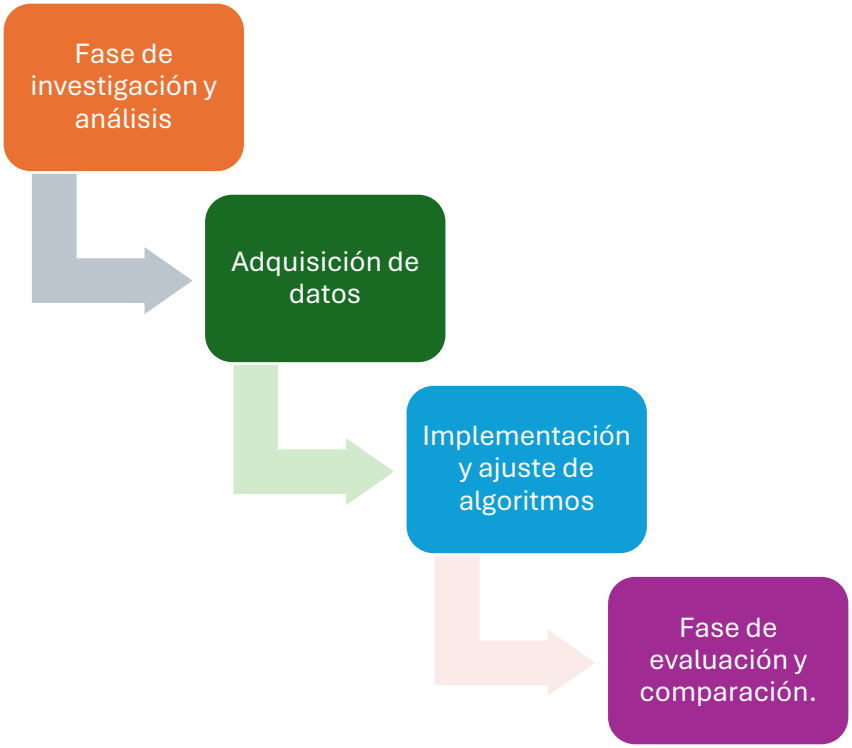
Para el desarrollo adecuado del proyecto de la evaluación del rendimiento de algoritmos de aprendizaje automático en la detección de spam para optimizar la calidad de la detección de correos electrónicos no deseados en un entorno de comunicación en línea, fue crucial seleccionar una metodología específica. Esta elección garantizó un proceso organizado y eficiente en el desarrollo del proyecto, asegurando una secuencia clara de ejecución con el fin de lograr el resultado deseado. Por ello, se seleccionó la metodología en Cascada, siguiendo las directrices del estándar IEEE 29148 para la especificación y gestión de requisitos en sistemas y software.

3.1.1. Metodología en Cascada

La metodología siguió un enfoque de desarrollo de proyectos en cascada, dividiendo el trabajo en fases secuenciales: fase de investigación y análisis, adquisición de datos, implementación y ajuste de algoritmos, y la fase de evaluación y comparación.

Figura 9

Metodología en cascada



Fuente. Autoría

La metodología siguió un enfoque cuantitativo, empleando un análisis de datos basado en medidas numéricas y estadísticas para evaluar el rendimiento de los algoritmos de aprendizaje automático en la detección de spam. Este método implicó la recolección de datos cuantificables, como métricas de precisión, exactitud, recall y F1 Score.

La precisión, fundamental en este contexto, indicó la proporción de clasificaciones correctas. El indicador de recall evaluó la capacidad del modelo para identificar todas las instancias relevantes, lo cual fue crucial para minimizar falsos negativos. La exactitud, otro indicador esencial, midió el porcentaje total de predicciones correctas sobre el número total de casos evaluados. Para equilibrar precisión y recall, se empleó el F1 Score, que ofreció una evaluación

ponderada en situaciones de desequilibrio entre clases o cuando los falsos positivos y negativos tuvieron un impacto significativo.

La investigación cuantitativa se centró en la recopilación y análisis de datos numéricos provenientes de la ejecución de los algoritmos en conjuntos de prueba específicos. Se utilizaron técnicas estadísticas para comparar y contrastar los resultados, proporcionando una evaluación objetiva y cuantificable del desempeño de cada algoritmo. Este enfoque permitió generalizar hallazgos a poblaciones más amplias y respaldar conclusiones basadas en evidencia numérica.

3.2. Requerimientos

La especificación y gestión de los requerimientos del proyecto se basaron en el estándar ISO/IEC/IEEE 29148:2018, el cual proporcionó un marco sólido para garantizar que los sistemas y software satisficieran las expectativas y necesidades de los stakeholders, además de los requerimientos del sistema y de la arquitectura. Este enfoque estandarizado permitió estructurar de manera clara los requerimientos esenciales para la implementación y evaluación de los algoritmos de aprendizaje automático en la detección de spam, abarcando aspectos técnicos, funcionales y operativos necesarios para optimizar la calidad y eficiencia del sistema.

Tabla 1

Acrónimos de requerimientos

Acrónimo	Descripción
StSR	Requerimientos de Stakeholders
SySR	Requerimientos del Sistema
SrSH	Requerimientos de Arquitectura, incluyendo Hardware y Software

3.2.1. Requerimientos de Stakeholders

Los requerimientos de los stakeholders reúnen las necesidades y expectativas de todos los involucrados en el proyecto, incluyendo clientes, usuarios y otros participantes clave. Estas demandas son cruciales para asegurar que el sistema desarrollado cumpla con los objetivos y especificaciones esperadas por todas las partes interesadas. En el contexto de evaluar el rendimiento de algoritmos de aprendizaje automático para la detección de spam, estas demandas incluyen desde las funcionalidades básicas hasta las características específicas de usabilidad y mantenimiento, garantizando así un sistema eficiente y eficaz que responda a las exigencias del entorno operativo.

Tabla 2

Requerimientos de Stakeholders

#	Requerimiento	Prioridad		
		Alta	**Media**	**Baja**
	Requerimientos operacionales			
StSR1	Soporte para almacenamiento en la nube para datos y modelos.	X		
StSR2	Comunicación eficiente entre la plataforma de procesamiento de datos y la interfaz gráfica.	X		
StSR3	Acceso seguro y rápido a los resultados	X		
StSR4	Escalabilidad para manejar un volumen creciente de correos.			X
	Requerimientos de usuarios			

StSR5	Interfaz gráfica amigable para visualización de resultados.		X	
StSR6	Facilidad de acceso a los datos y resultados.	X		
StSR7	Documentación y soporte técnico accesible.		X	
StSR8	Personalización de reportes y dashboards.			X
StSR9	Funcionalidades de ayuda y tutoriales integrados.			X

3.2.2. Requerimientos del Sistema

Los requerimientos del sistema especificaron las funcionalidades, limitaciones y comportamientos que el sistema debía tener para satisfacer las expectativas de los stakeholders y los objetivos del proyecto. Estos requisitos incluyeron aspectos técnicos como la facilidad de uso, la interoperabilidad, el rendimiento y la seguridad del sistema. En el contexto de evaluar el rendimiento de algoritmos de aprendizaje automático para la detección de spam, estos requerimientos garantizaron que las herramientas y tecnologías utilizadas se integraran de manera coherente y eficiente, permitiendo un análisis y detección de spam óptimos en el servidor de correo.

Tabla 3

Requerimientos del Sistema

#	Requerimiento	Prioridad		
		Alto	**Media**	**Bajo**
	Requerimientos de interfaz			

SySR1	Integración con sistemas de autenticación y seguridad.	X		
SySR2	Compatibilidad con APIs externas para tratamiento de datos.			X
	Requerimientos de uso			
SySR3	Sistema fácil de manejar para los administradores.	X		
SySR4	Acceso a resultados y parámetros de análisis.			
SySR5	Configuración flexible de parámetros de detección de spam.			X
	Requerimientos de performance			
SySR6	Alta precisión en la detección de spam.	X		
SySR7	Bajo consumo de recursos del servidor.		X	
SySR8	Almacenamiento y procesamiento de datos eficiente.	X		
SySR9	Integración con herramientas de tratamiento de datos.	X		
SySR10	Visualización gráfica de resultados en tiempo real.	X		
SySR11	Tiempo de respuesta bajo al procesar grandes volúmenes de datos.			X
	Requerimientos de modo y estado			

SySR12	La interfaz gráfica permite visualización detallada de cada correo.	X	
SySR13	Soporte para múltiples idiomas en la interfaz gráfica.		X
	Requerimientos físicos		
SySR14	Hardware del servidor ubicado en un ambiente seguro y controlado.	X	
SySR15	Protección adecuada del hardware contra fallos eléctricos.	X	
SySR16	Suficiente espacio de almacenamiento para datos y modelos.	X	
SySR17	Mantenimiento regular del hardware para asegurar su longevidad.		X

3.2.3. Requerimientos de Arquitectura

Los requerimientos de arquitectura establecieron los componentes esenciales tanto de hardware como de software que eran necesarios para el funcionamiento e implementación efectiva del sistema. Estos requerimientos garantizaron que todos los elementos tecnológicos, desde las plataformas de procesamiento y almacenamiento hasta las herramientas de programación y los dispositivos de soporte, fueran compatibles y funcionaran de manera integrada. En el contexto de la evaluación del rendimiento de algoritmos de aprendizaje automático para la detección de spam, los requerimientos de arquitectura aseguraron que la infraestructura tecnológica soportara

eficientemente las operaciones del sistema, proporcionando una base sólida y robusta para el desarrollo y despliegue de los algoritmos.

Tabla 4

Requerimientos de Arquitectura

#	Requerimiento	Prioridad		
		Alta	**Media**	**Baja**
	Requerimientos de diseño			
SrSH1	Hardware y software compatibles entre sí.	X		
SrSH2	Comunicación eficiente entre la plataforma de procesamiento de datos y la interfaz gráfica.	X		
SrSH3	Sistema accesible y manejable por los administradores e integración con los algoritmos de detección de spam	X		
SrSH4	Capacidad de modificar y personalizar el flujo de trabajo.			X
	Requerimientos lógicos			
SrSH5	Acceso rápido y seguro a la base de datos.	X		
SrSH6	Compatibilidad con sistemas operativos Linux.	X		
SrSH7	Compatibilidad con múltiples lenguajes de programación.	X		
SrSH8	Compatibilidad con herramientas de análisis de logs.			X
	Requerimientos de hardware			

SrSH9	Servidor con al menos 8 GB de RAM y procesador multicore.	X		
SrSH10	Almacenamiento SSD para rápido acceso a datos.	X		
SrSH11	Conexión a internet estable y rápida.	X		
SrSH12	Suficiente capacidad de almacenamiento (mínimo 1 TB).	X		
SrSH13	Hardware de respaldo para garantizar la continuidad del servicio.		X	
	Requerimientos de software			
SrSH14	Compatibilidad con librerías de aprendizaje automático como Scikit-learn.	X		
SrSH15	Licencias Open Source para todas las herramientas de software utilizadas.	X		
SrSH16	Software de tratamiento de datos y visualización como Pandas y Matplotlib.	X		
SrSH17	IDE de programación compatible y eficiente.	X		
SrSH18	Facilidad de uso y colaboración en tiempo real.	X		
SrSH19	Capacidad para ejecutar código en la nube.		X	
SrSH20	Soporte para herramientas de CI/CD.			X
SrSH21	Capacidad de integrar herramientas de monitoreo de rendimiento.			X

3.3. Comparación de los Algoritmos de Aprendizaje Automático

En esta etapa se recolectó la información y se documentaron las características que influyeron en la comparación de los algoritmos de aprendizaje automático. Para este apartado se elaboró un informe correspondiente al análisis de estos criterios, el cual se documentó en el ANEXO C.

En el anexo, se especificaron las características que cada algoritmo debía cumplir para ser considerado adecuado en la detección de spam. A continuación, en la Tabla 4, que fue extraída del ANEXO C, se detalló cómo cada uno de los algoritmos seleccionados cumplió con estos requisitos.

Tabla 5

Comparación de los Algoritmos de Aprendizaje Automático

Características	KNN	Naive Bayes	Decision Tree	Logistic Regression	SVM	Random Forest	Redes Neuronales
Simplicidad e Intuición	X	X	X	X	-	-	-
Eficiencia para grandes datasets	-	X	-	-	-	X	X
Manejo de Datos Categóricos	X	X	X	X	-	X	X
Manejo de Datos Continuos	X	-	X	X	X	X	X
Facilidad de Interpretación	X	-	X	-	-	-	-

Baja Complejidad Computacional	-	X	-	-	-	-	-
Reducción de Sobreajuste	-	X	-	-	X	-	-

Cumple “X”, No cumple “-”

Los algoritmos K-Nearest Neighbors (KNN), Naive Bayes y Decision Tree fueron seleccionados por su desempeño en múltiples criterios relevantes para la detección de spam. KNN destacó por su simplicidad y manejo de datos continuos y categóricos, aunque pudo ser computacionalmente intensivo. Naive Bayes fue altamente eficiente y manejó bien grandes datasets y datos categóricos, con baja complejidad computacional, pero su supuesta independencia entre características pudo ser una limitación. Decision Tree fue fácil de interpretar y manejó tanto datos categóricos como continuos, aunque pudo ser propenso al sobreajuste y no siempre eficiente con grandes datasets. Estos algoritmos combinaron simplicidad, eficiencia y capacidad de interpretación, proporcionando un buen equilibrio para un sistema de detección de spam robusto y eficiente.

3.4. Elección del lenguaje de programación

La elección del lenguaje de programación fue fundamental para asegurar un desarrollo eficiente y efectivo en un sistema de detección de spam mediante algoritmos de aprendizaje automático. En este análisis, se evaluaron ocho lenguajes de programación: Python, R, Julia, MATLAB, Java, Scala, JavaScript (con TensorFlow.js) y C++. La decisión se basó en la consideración de varios requerimientos clave, como la compatibilidad con librerías de aprendizaje automático como Scikit-learn, TensorFlow y PyTorch (SrSH14), la disponibilidad de licencias Open Source para todas las herramientas de software utilizadas (SrSH15), y la

integración con software de tratamiento de datos y visualización como Pandas y Matplotlib (SrSH16). Además de otras características detalladas en el ANEXO D. La tabla a continuación mostró cómo cada lenguaje cumplió con estos requerimientos:

Tabla 6

Elección del Lenguaje de Programación

Lenguaje de	**Requisitos**						
Programación	**SrSH14**	**SrSH15**	**SrSH16**	**SrSH17**	**SrSH18**	**SrSH19**	**Valor**
Python	1	1	1	1	1	1	6
R	0	1	1	1	0	0	3
Julia	0	1	0	1	0	1	3
MATLAB	0	1	1	1	0	1	4
Java	1	0	1	0	0	1	3
Scala	0	0	0	0	0	1	1
JavaScript (TensorFlow.js)	0	1	1	0	0	1	3
C++	0	0	0	0	1	0	1

Cumple “1”, No cumple “0”

De acuerdo con la Tabla 6, Python fue elegido el lenguaje de programación principal debido a su capacidad para cumplir con la mayoría de los requerimientos de software clave. Python destacó por su amplia compatibilidad con librerías de aprendizaje automático como Scikit-learn, TensorFlow y PyTorch, lo que lo hizo ideal para desarrollar sistemas de detección de spam. Además, su integración con herramientas de tratamiento de datos y visualización como Pandas y Matplotlib aseguró un manejo eficiente de grandes volúmenes de datos, mientras que

su soporte para herramientas de CI/CD facilitó la implementación continua y el monitoreo del rendimiento. Estas características hicieron de Python la opción más completa y eficiente para este proyecto.

3.5. Elección del IDE de programación

La elección del entorno de desarrollo integrado (IDE) fue crucial para facilitar el desarrollo y la implementación de los algoritmos de detección de spam, especialmente considerando que el lenguaje de programación elegido fue Python. Este enfoque aseguró la compatibilidad con librerías avanzadas de aprendizaje automático y herramientas de visualización de datos. En este estudio, se evaluaron los IDEs más populares para trabajar con Python: Google Colab, Jupyter Notebook, Visual Studio Code, PyCharm, Spyder y Anaconda.

Se consideraron varios requerimientos clave, como la compatibilidad y eficiencia del IDE (SrSH17), el acceso a resultados y parámetros de análisis (SySR4), la compatibilidad con múltiples lenguajes de programación y extensiones (SrSH7), el soporte para almacenamiento en la nube para datos y modelos (StSR1), y la facilidad de acceso a los datos y resultados (StSR6). La tabla a continuación mostró cómo cada IDE cumplió con estos requerimientos:

Tabla 7

Elección del Entorno de Desarrollo Integrado (IDE)

IDE de Programación	**Requisitos**					
	SrSH17	**SySR4**	**SrSH7**	**StSR1**	**StSR6**	**Valor**
Google Colab	1	1	1	1	1	5
Jupyter Notebook	1	1	1	0	1	4
Visual Studio Code	1	1	0	0	1	3
PyCharm	1	0	1	0	1	3

Spyder	1	0	1	0	1	3
Anaconda	0	0	1	1	0	2

Cumple “1”, No cumple “0”

Con base en la Tabla 7, Google Colab fue elegido el entorno de desarrollo principal debido a sus numerosas ventajas: su compatibilidad y eficiencia como IDE, la capacidad de acceder a resultados y parámetros de análisis de manera rápida y sencilla, y su integración con almacenamiento en la nube para datos y modelos. Además, Google Colab facilitó el acceso a los datos y resultados y ofreció un entorno ideal para ejecutar código de forma eficiente, especialmente en proyectos de machine learning que requerían recursos computacionales avanzados.

3.6. Elección del Servidor de Correo Electrónico

La elección del servidor de correo electrónico fue crucial para garantizar un entorno de prueba eficiente y robusto para la implementación de los algoritmos de detección de spam. En este estudio, se evaluaron diferentes servidores de correo: Poste.io, Zimbra y Mailcow, Microsoft Exchange y Open-Xchange. Se consideraron varios requerimientos clave, incluyendo la integración con sistemas de autenticación y seguridad (SrSR1), la comunicación eficiente entre la plataforma de procesamiento de datos y la interfaz gráfica (SrSH2), el acceso rápido y seguro a la base de datos (SrSH5), la integración con los algoritmos de detección de spam (SrSH3), y el soporte para almacenamiento en la nube para datos y modelos (StSR1). La tabla a continuación mostró cómo cada servidor cumplió con estos requerimientos:

Tabla 8

Elección del Servidor de Correo Electrónico

Servidor de Correo	Requisitos					
	SySR1	**SrSH2**	**SrSH3**	**SrSH5**	**StSR1**	**Valor**
Poste.io	1	0	1	1	1	4
Zimbra	1	1	1	1	1	5
Mailcow	1	1	0	1	0	3
Microsoft Exchange	1	1	0	1	1	4
Open-Xchange	1	0	0	0	0	1
ProtonMail	1	1	1	0	1	4

Cumple “1”, No cumple “0”

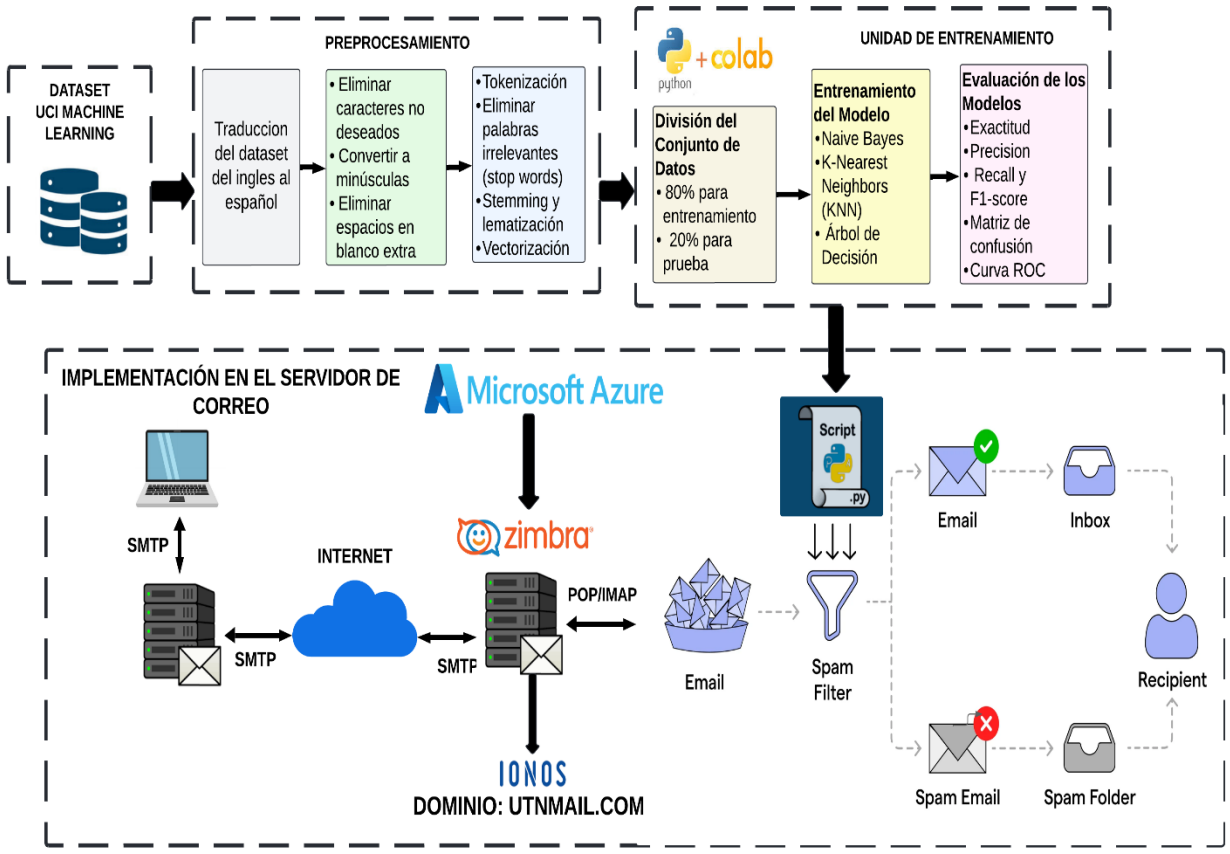
Zimbra fue elegido el servidor de correo principal debido a sus numerosas ventajas. Su sólida integración con sistemas de autenticación y seguridad aseguró una protección robusta para los datos y usuarios. La comunicación eficiente entre la plataforma de procesamiento de datos y la interfaz gráfica y el acceso rápido y seguro a la base de datos permitieron un manejo fluido y eficaz de la información. Además, Zimbra se destacó por su integración con los algoritmos de detección de spam y su soporte para almacenamiento en la nube para datos y modelos, lo que lo convirtió en una opción ideal para mantener un entorno de prueba eficiente y bien gestionado.

3.7. Diagrama de Bloques del Sistema

El esquema ayudó a visualizar cómo las diferentes herramientas y tecnologías se integraron y colaboraron para lograr un análisis eficaz y una detección precisa en el servidor de correo.

Figura 10

Diagrama de bloques del sistema



El esquema estuvo dividido en bloques como: bloque de Dataset UCI Machine Learning, bloque de preprocesamiento, bloque de entrenamiento y bloque de implementación en el servidor de correo.

- **Bloque de Dataset UCI Machine Learning:** representó la fuente de datos inicial utilizada para el análisis. Este conjunto de datos fue crucial para entrenar y evaluar los algoritmos de detección de spam.
- **Bloque de Preprocesamiento:** se llevaron a cabo varias etapas clave. Primero, se realizó la traducción del dataset del inglés al español. Luego, se procedió a la eliminación de caracteres no deseados, conversión a minúsculas y eliminación de

espacios en blanco extra, asegurando que los datos estuvieran limpios y normalizados. Posteriormente, se efectuaron procesos de tokenización, eliminación de palabras irrelevantes (stop words), stemming y lematización, y vectorización, transformando así los datos en una forma adecuada para el entrenamiento del modelo.

- **Bloque de Entrenamiento:** se encargó de la división del conjunto de datos, utilizando el 80% para entrenamiento y el 20% restante para prueba. En esta fase, se entrenaron varios modelos de aprendizaje automático, incluyendo Naive Bayes, K-Nearest Neighbors (KNN) y Árbol de Decisión. La evaluación de estos modelos se realizó mediante métricas como exactitud, precisión, recall y F1-score, además de utilizar matrices de confusión y curvas Rho para una valoración completa del rendimiento.
- **Bloque de Implementación en el Servidor de Correo**: durante la implementación, se demostró el flujo completo de correos electrónicos desde el servidor de origen hasta el destinatario final. Este flujo incluyó el paso por un filtro de spam que utilizó algoritmos de clasificación para identificar y separar los correos legítimos de los no deseados. Los correos detectados como spam fueron redirigidos automáticamente a la carpeta de spam, garantizando así una filtración y entrega de mensajes eficiente y segura dentro del sistema de correo. Esto aseguró no solo la seguridad y limpieza de la bandeja de entrada del usuario, sino también la estabilidad y confiabilidad del servidor de correo en general.

3.8. Diagrama de Flujo

El diagrama de flujo del sistema para la detección de spam mediante algoritmos de aprendizaje automático ilustró de manera clara y secuencial los pasos involucrados en el proceso de clasificación de mensajes. Este diagrama se compuso de varios bloques interrelacionados que permitieron visualizar cómo los datos fluyeron desde su captura hasta la clasificación final. A

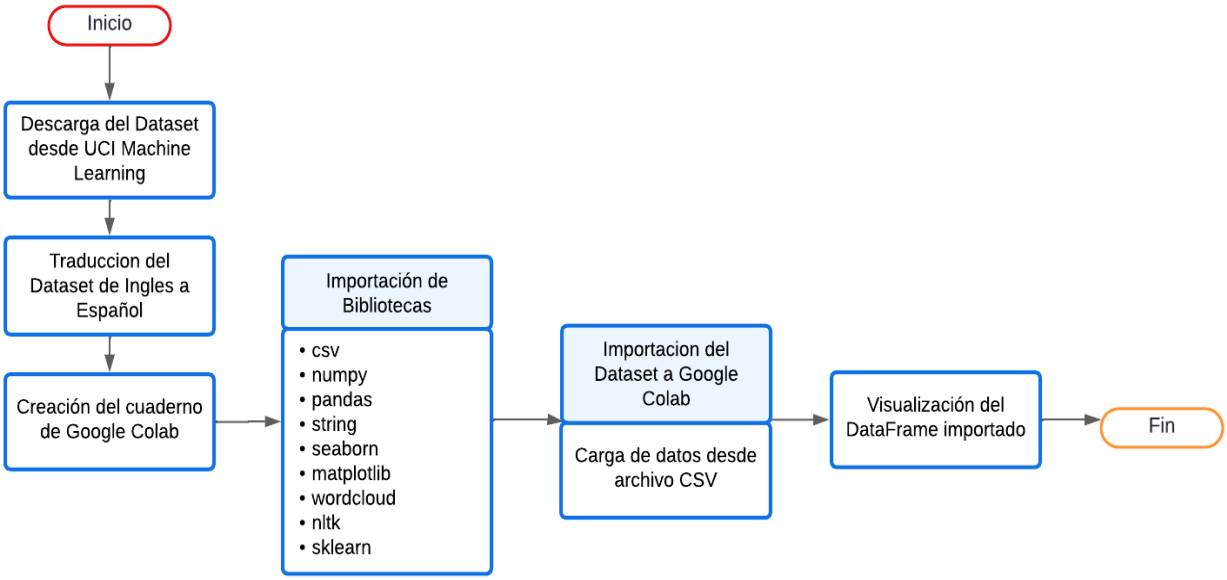
continuación, se describieron los bloques principales que componían el proceso de detección de spam utilizando modelos de aprendizaje automático, cada uno de los cuales desempeñó una función esencial en la identificación precisa de los correos no deseados.

3.8.1. Flujograma de Adquisición y Carga de Datos:

El flujograma comenzó con la identificación y selección del dataset adecuado desde UCI Machine Learning Repository. Después, se procedió a la descarga del dataset, seguida de la traducción del contenido del dataset de inglés a español. Luego se realizó la importación de las bibliotecas necesarias, tales como csv, numpy, pandas, string, seaborn, matplotlib, wordcloud, nltk, y sklearn. Posteriormente, el dataset fue cargado en el entorno de trabajo, el cual era Google Colab, utilizando pandas para su manipulación.

Figura 11

Flujograma de adquisición y carga de datos

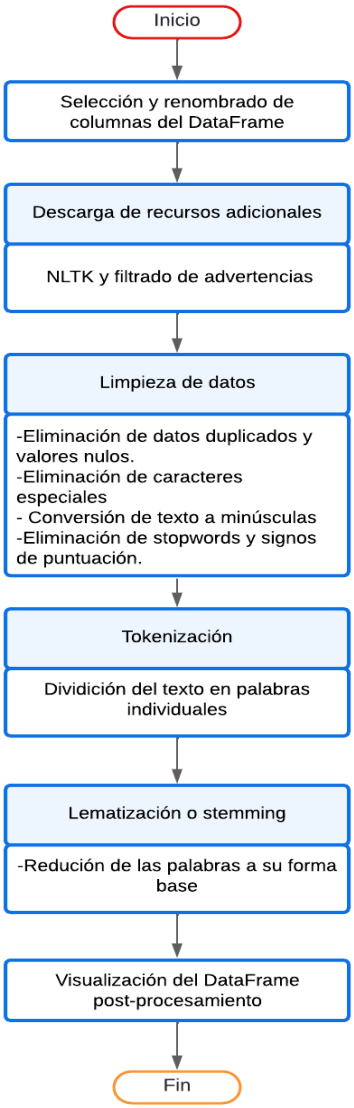


Durante el preprocesamiento de datos, se inició con la selección y renombrado de columnas del DataFrame, donde se eligieron las columnas pertinentes y se les asignaron nombres adecuados para facilitar el análisis subsiguiente. Luego, se procedió con la descarga de recursos

adicionales desde NLTK (Natural Language Toolkit) y el filtrado de advertencias para asegurar un proceso sin interrupciones. Se continuó con la tokenización de mensajes y la eliminación de stopwords y signos de puntuación, optimizando así el texto para reducir ruido y mejorar la precisión del análisis. Finalmente, se mostró una visualización inicial del DataFrame post-procesamiento, ofreciendo una vista preliminar de los datos listos para la siguiente fase.

Figura 12

Flujograma de preprocesamiento de datos

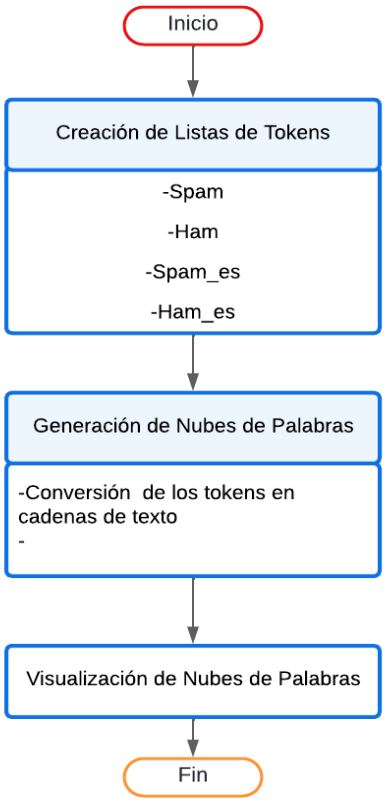


3.8.2. Flujograma de Tokenización y Visualización de Nubes de Palabras

En la Tokenización y Visualización, se realizó un proceso clave para el análisis de datos textuales. Comenzó con la creación de listas de tokens para diferentes categorías, tales como spam, ham, spam_es y ham_es. Posteriormente, estos tokens se convirtieron en cadenas de texto y se utilizaron para generar nubes de palabras específicas para cada categoría en el siguiente paso, etiquetado como "Generación de Nubes de Palabras". Estas nubes de palabras visualizaron las palabras más frecuentes en cada conjunto de datos, proporcionando una representación visual intuitiva de las características distintivas de los mensajes spam y no spam en los dos idiomas analizados. Finalmente, se concluyó con la visualización de las nubes de palabras generadas, lo que facilitó la interpretación y comprensión de los patrones textuales identificados en el análisis.

Figura 13

Flujograma de tokenización y visualización de nube de palabras

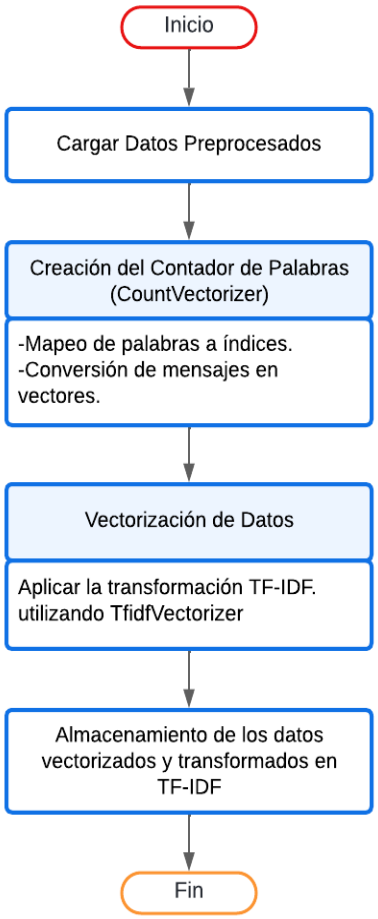


3.8.3. Flujograma de Vectorización y TF-IDF

En la vectorización y TF-IDF (Term Frequency-Inverse Document Frequency), se comenzó cargando los datos preprocesados, seguidamente se creó un contador de palabras y se ordenaron por frecuencia. A continuación, se mapearon las palabras a índices y se convirtieron los mensajes en vectores. Luego, se procedió a la vectorización de los datos utilizando TfidfVectorizer. Finalmente, se guardaron los datos vectorizados y transformados en TF-IDF para su uso en la fase de entrenamiento.

Figura 14

Flujograma de vectorización y TF-IDF

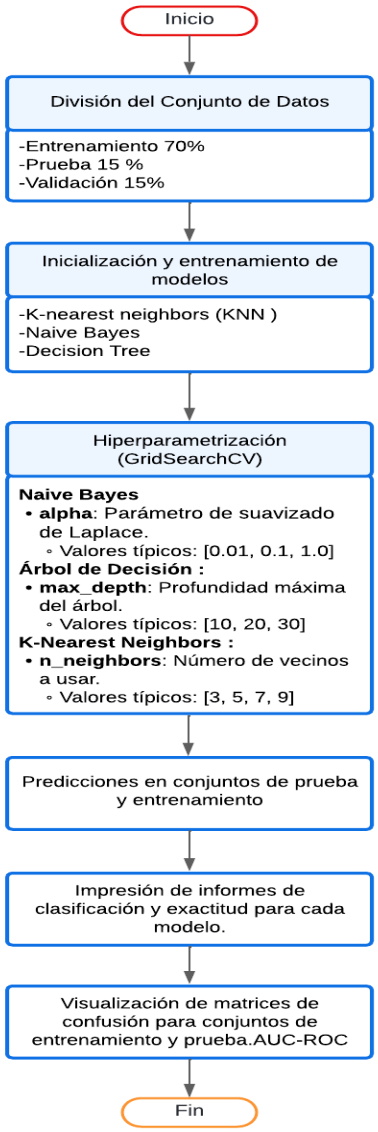


3.8.4. Flujograma del Entrenamiento y Evaluación de Modelos:

El flujograma del entrenamiento y evaluación de modelos comenzó con la división de los datos en conjuntos de entrenamiento, prueba y validación. Se configuraron los parámetros y se realizó una búsqueda exhaustiva de hiperparámetros utilizando GridSearchCV. Los modelos se entrenaron con los datos de entrenamiento y se evaluaron utilizando métricas como AUC-ROC, F1-score y matriz de confusión.

Figura 15

Flujograma de entrenamiento y evaluación de modelos

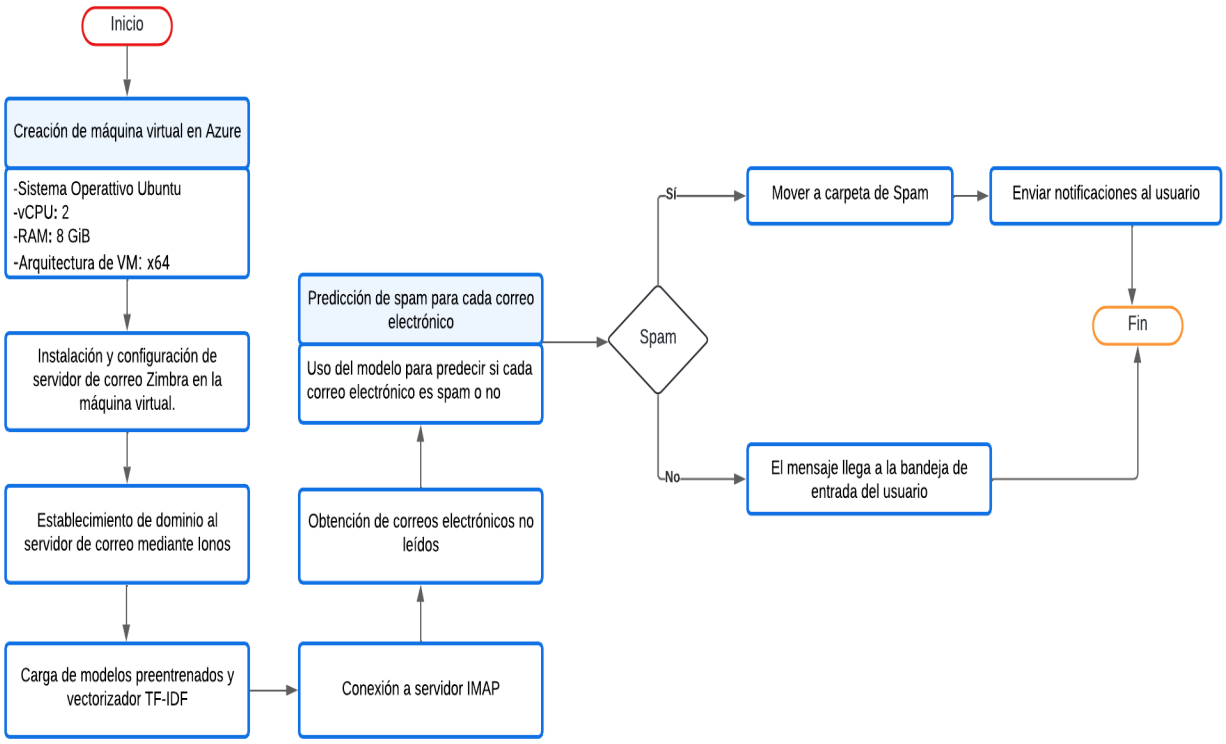


3.8.5. Flujograma de Implementación y Predicción

En la implementación y predicción en tiempo real, se configuró una máquina virtual en Azure utilizando Ubuntu y se levantó el servidor de Poste.io a través de Docker. Luego, se estableció un dominio al servidor de correo mediante Ionos. Se cargaron los modelos preentrenados y el vectorizador TF-IDF. A continuación, se conectó al servidor IMAP para obtener correos electrónicos no leídos. Para cada correo electrónico, se realizó una predicción de spam. Finalmente, se tomaron acciones posteriores, como mover el correo a la carpeta de spam o enviar notificaciones por correo.

Figura 16

Flujograma de implementación y predicción



En esta sección se detalló el proceso de preparación de los datos utilizados para entrenar y probar los algoritmos de detección de spam. Los datos fueron cuidadosamente seleccionados y sometidos a diversas técnicas de preprocesamiento para garantizar su adecuación y calidad para

el análisis. A continuación, se describió el origen de los datos y las técnicas específicas de preprocesamiento empleadas.

3.8.6. Origen de los Datos

En este proyecto, los datos utilizados para entrenar y probar los algoritmos de detección de spam se obtuvieron del repositorio de Machine Learning de la Universidad de California, Irvine (UCI). Este conjunto de datos fue ampliamente utilizado en la comunidad de aprendizaje automático para la investigación y evaluación de algoritmos de detección de spam.

El conjunto de datos consistió en una colección de mensajes etiquetados como spam o no spam, lo que proporcionó una base sólida para entrenar y evaluar los algoritmos de detección. Cada mensaje en el conjunto de datos estuvo representado por un conjunto de características relevantes, como el contenido del mensaje, la frecuencia de ciertas palabras clave, la longitud del mensaje, entre otros.

Figura 17

Dataset recopilado del repositorio de UCI Machine Learning

v1	v2
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv
ham	Even my brother is not like to speak with me. They treat me like aids patent.
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune
spam	WINNER!! As a valued network customer you have been selected to receivea å£900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.
spam	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030
ham	I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.
spam	SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ TsandCs apply Reply HL 4 info
spam	URGENT! You have won a 1 week FREE membership in our å£100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18
ham	I've been searching for the right words to thank you for this breather. I promise i wont take your help for granted and will fulfil my promise. You have been wonderful and a blessing at all times.
ham	I HAVE A DATE ON SUNDAY WITH WILL!!
spam	XXXMobileMovieClub: To use your credit, click the WAP link in the next txt message or click here>> http://wap. xxxmobilemovieclub.com?n=QJKGIGHJJGCBL
ham	Oh k...i'm watching here:)
ham	Eh u remember how 2 spell his name... Yes i did. He v naughty make until i v wet.

Nota. Conjunto de datos recopilados del repositorio de la Universidad de California, Irvine (UCI) etiquetados como spam y no spam.

3.8.6.1. Expansión del Conjunto de Datos

El dataset original del repositorio de UCI Machine Learning se adaptó para la detección de spam en español mediante la traducción de mensajes en inglés, permitiendo a los modelos reconocer patrones en ambos idiomas. Sin embargo, no todos los mensajes fueron utilizados, ya que algunos perdían su contexto o efectividad debido a expresiones sin equivalentes directos en español. Solo se seleccionaron aquellos cuya traducción mantuvo el significado original, asegurando coherencia y efectividad en la clasificación de spam en entornos hispanohablantes.

La traducción se justifica porque países como Estados Unidos, China y Vietnam son los principales emisores de spam (Moreno, 2018), y muchos de estos correos están en inglés. Adaptarlos al español mejora la detección y adecuación de los modelos para usuarios hispanohablantes. Contar con un experto en traducción habría optimizado aún más la precisión del proceso.

Figura 18

Dataset modificado a partir del repositorio de UCI Machine Learning

ham	Ard 6 like dat lor.
ham	Why don't you wait 'til at least wednesday to see if you get your .
ham	Huh y lei...
spam	REMINDER FROM O2: To get 2.50 pounds free call credit and details of great offers pls reply 2 this text with your valid name, house no and postcode
spam	This is the 2nd time we have tried 2 contact u. U have won the å£750 Pound prize. 2 claim is easy, call 087187272008 NOW1! Only 10p per minute. BT-national-rate.
ham	Will Ì_ b going to esplanade fr home?
ham	Pity, * was in mood for that. So...any other suggestions?
ham	The guy did some bitching but I acted like i'd be interested in buying something else next week and he gave it to us for free
ham	Rofl. Its true to its name
ham_es	Anda hasta el punto de Jurong, ¡estás loco! Disponible solo en Bugis y en el buffet de Great World... Hay amor allí, ¿verdad? Mensaje traducido al español del dataset original
ham_es	Está bien... Solo bromeaba contigo...
spam_es	Entrada gratuita en una competición semanal para ganar entradas para la final de la Copa FA el 21 de mayo de 2005. Envía FA al 87121 para recibir la pregunta de entrada (tarifa es
ham_es	Tú no lo digas tan pronto, ¿eh? Ya lo ves y luego lo dices...
ham_es	No creo que él vaya a USF, aunque vive por aquí cerca.
spam_es	FreeMsg ¡Hola cariño! ¡Han pasado 3 semanas y aún no he tenido noticias tuyas! ¿Te apetece divertirte todavía? Estoy disponible. Coste estándar para enviar. £1.50 para recibir.

Nota. Conjunto de datos modificado a partir del repositorio de la Universidad de California, Irvine (UCI), con mensajes traducidos al español y etiquetados como spam y no spam. Se excluyeron aquellos cuya traducción no conservaba el contexto original.

3.8.7. Técnicas de Preprocesamiento Utilizadas

Para el preprocesamiento de los mensajes, se aplicaron técnicas como la eliminación de signos de puntuación, tokenización, eliminación de stopwords, lematización o stemming y vectorización. Estas transformaciones permitieron estructurar el texto de manera más eficiente para su análisis y clasificación por los algoritmos de detección de spam.

3.8.7.1. Eliminación de signos de puntuación.

La eliminación de los signos de puntuación es una técnica esencial en el preprocesamiento de texto, especialmente en el análisis de datos para sistemas de detección de spam. Esta técnica implica eliminar todos los caracteres de puntuación como comas, puntos, signos de interrogación, signos de exclamación, paréntesis y otros símbolos similares que no aportan información semántica relevante para los algoritmos de aprendizaje automático (Arengas et al., 2024).

Los signos de puntuación pueden añadir ruido a los datos textuales, dificultando el proceso de análisis y la reducción del texto a sus componentes más fundamentales. Al eliminar estos caracteres, se facilita la tokenización del texto, es decir, la división del texto en unidades más pequeñas, como palabras o frases, que pueden ser analizadas individualmente. Además, eliminar los signos de puntuación contribuye a la normalización de los datos, asegurando que las palabras se consideren en su forma más básica y directa.

Este proceso también es crucial para asegurar la consistencia en la representación de los datos. Diferentes usuarios pueden utilizar distintos signos de puntuación de maneras diversas, lo que podría llevar a interpretaciones inconsistentes si no se gestionan adecuadamente. Al eliminar estos signos, se reduce la variabilidad y se mejora la precisión y la eficiencia del modelo de aprendizaje automático.

3.8.7.2. Tokenización de los mensajes de texto

La tokenización en el procesamiento del lenguaje natural es un paso crucial que implica dividir el texto en unidades más pequeñas, conocidas como tokens, que pueden ser palabras, frases o incluso caracteres individuales, dependiendo del contexto y la tarea específica. Esta etapa es esencial para diversas aplicaciones de PLN, como el análisis de sentimientos, la traducción automática y la generación de texto (Vijayarani & Janani, 2016).

La tokenización permite que los algoritmos de PLN trabajen con fragmentos discretos de texto, lo que facilita la extracción de características y el análisis semántico. Al dividir el texto en tokens, se crea una base para la implementación de técnicas más avanzadas, como el etiquetado de partes del discurso, la lematización y la eliminación de palabras irrelevantes.

Por ejemplo, en la frase "El perro marrón saltó sobre el muro", la tokenización podría generar tokens individuales como "El", "perro", "marrón", "saltó", "sobre", "el", "muro", los cuales pueden ser posteriormente procesados y analizados por el algoritmo para comprender el significado de la frase.

3.8.7.3. Eliminación de stopwords

Eliminar las stopwords es una técnica esencial en el preprocesamiento de texto para sistemas de detección de spam. Estas stopwords son palabras comunes que aparecen con frecuencia en el lenguaje natural, como "el", "la", "y", "de", entre otras, pero que no tienen un significado semántico relevante por sí solas para el análisis del contenido.

Al quitar las stopwords del texto, se reduce el ruido y se centra el análisis en las palabras que transmiten información importante sobre el contenido del mensaje. Esto mejora la eficacia de los algoritmos de aprendizaje automático al disminuir la cantidad de datos y aumentar la precisión de las predicciones (Cole et al., 2019).

Por ejemplo, en un mensaje como "El gato está en la casa", las stopwords "el" y "en" no añaden información relevante al contexto del mensaje. Al eliminarlas, el mensaje se simplifica a "gato está casa", lo que facilita la identificación de patrones significativos para detectar spam.

Además de mejorar la eficacia del análisis, la eliminación de stopwords también contribuye a la normalización del texto, asegurando que las palabras consideradas sean aquellas que tienen un mayor impacto en la interpretación del mensaje. Además, reduce el tiempo de procesamiento y la complejidad computacional al eliminar palabras innecesarias que no contribuyen al análisis final.

3.8.7.4. Lematización o stemming

La lematización o el stemming son técnicas cruciales en el preprocesamiento de texto para sistemas de detección de spam. Ambas tienen como objetivo simplificar las palabras a su forma base o raíz, lo que ayuda a estandarizar el texto y agrupar palabras relacionadas bajo una única forma. Sin embargo, se diferencian en su enfoque y nivel de complejidad.

La lematización es un proceso más avanzado que busca reducir las palabras a su lema, es decir, su forma canónica en el idioma. Por ejemplo, palabras como "corriendo" y "corrió" se transformarían en su lema "correr". Aunque este método es más preciso, también es más complejo desde el punto de vista computacional, ya que requiere el uso de recursos lingüísticos como diccionarios y reglas gramaticales (Huet, 2024).

Por otro lado, el stemming es un proceso más sencillo que elimina los sufijos y prefijos de las palabras para obtener su raíz. Por ejemplo, palabras como "corriendo" y "corrió" se reducirían a la raíz "corr". Aunque menos preciso que la lematización, el stemming es más rápido y menos exigente computacionalmente, lo que lo hace más adecuado para aplicaciones donde la eficiencia es prioritaria.

Tanto la lematización como el stemming ayudan a reducir la complejidad del texto y agrupar palabras similares bajo una misma forma, facilitando el análisis y la detección de patrones relevantes para identificar spam. Estas técnicas también contribuyen a la normalización del texto, garantizando que las palabras se consideren en su forma más básica y directa, independientemente de sus variaciones gramaticales.

3.8.7.5. Vectorización

La vectorización es una técnica esencial en el preprocesamiento de texto para sistemas de detección de spam. Implica convertir el texto en una forma numérica comprensible y procesable por los algoritmos de aprendizaje automático.

Hay varios enfoques de vectorización, siendo los más comunes la representación de bolsa de palabras (Bag of Words) y el modelo TF-IDF (Term Frequency-Inverse Document Frequency).

En la representación de bolsa de palabras, cada documento de texto se transforma en un vector donde cada componente corresponde a una palabra única del vocabulario. La matriz resultante, llamada matriz de términos-documento, muestra la frecuencia de cada palabra en cada documento.

Por otro lado, el modelo TF-IDF asigna un peso a cada palabra basado en su frecuencia en el documento y en todo el corpus. Las palabras que aparecen mucho en un documento, pero raramente en el corpus tendrán un peso más alto, resaltando así los términos más importantes y reduciendo la influencia de las palabras comunes que no aportan información relevante (Cole et al., 2019).

La vectorización permite representar el texto de manera estructurada y cuantitativa, facilitando su procesamiento por parte de los algoritmos de aprendizaje automático. Además,

reduce la dimensionalidad de los datos y ayuda a identificar patrones y características importantes para la detección de spam.

3.9. Implementación

La implementación del sistema de detección de spam se realizó a través de varias fases que incluyeron la importación y carga de datos, preprocesamiento, vectorización, entrenamiento y evaluación de modelos. Cada fase fue cuidadosamente desarrollada para garantizar la adecuada preparación y procesamiento de los datos, empleando técnicas avanzadas de procesamiento de lenguaje natural y algoritmos de aprendizaje automático.

La configuración de la infraestructura se llevó a cabo en una máquina virtual en Azure, donde se desplegó el servidor de correo Zimbra. Además, se estableció un dominio utilizando Ionos para gestionar los correos electrónicos. Finalmente, se integraron los modelos preentrenados en el sistema, que se conectó a un servidor IMAP para permitir la predicción en tiempo real de los correos electrónicos, asegurando así un funcionamiento eficiente y preciso del sistema de detección de spam.

3.9.1. Fase de Importación y Carga de Datos

En esta fase inicial del proyecto, se abordaron dos tareas cruciales: la importación de bibliotecas necesarias y la carga del dataset. Estas tareas fueron fundamentales para preparar el entorno de trabajo y los datos que se utilizarían en las siguientes etapas del análisis y la construcción de modelos de aprendizaje automático.

3.9.1.1. Importación de Bibliotecas

El primer paso en cualquier proyecto de análisis de datos o desarrollo de modelos de aprendizaje automático fue importar las bibliotecas necesarias. Estas bibliotecas proporcionaron

las herramientas y funciones requeridas para manipular, visualizar y analizar datos, así como para construir y evaluar modelos.

Figura 19

Importación de bibliotecas

```
# Importar librerias
import csv  # Importa el módulo csv para trabajar con archivos CSV
import numpy as np  # Importa NumPy para operaciones numéricas eficientes
import pandas as pd  # Importa Pandas para manipulación y análisis de datos
import string  # Importa el módulo string para manipulaciones de cadenas de texto
import seaborn as sns  # Importa Seaborn para visualización de datos estadísticos
import matplotlib.pyplot as plt  # Importa Matplotlib para visualización de gráficos
from wordcloud import WordCloud  # Importa WordCloud para visualizar nubes de palabras
import nltk  # Importa NLTK (Natural Language Toolkit) para procesamiento de lenguaje natural
from nltk.corpus import stopwords  # Importa el corpus de stopwords de NLTK
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer  # Importa clases para vectorización de texto
from sklearn.tree import DecisionTreeClassifier  # Importa el clasificador de árbol de decisión de sklearn
from sklearn.model_selection import GridSearchCV, train_test_split, StratifiedKFold, cross_val_score, learning_curve
from sklearn.metrics import roc_auc_score, f1_score, confusion_matrix  # Importa métricas de evaluación de modelos
from sklearn.naive_bayes import MultinomialNB  # Importa el clasificador Naive Bayes multinomial de sklearn
from sklearn.neighbors import KNeighborsClassifier  # Importa el clasificador KNN de sklearn
```

Estas bibliotecas ofrecieron capacidades para manipulación de datos, visualización, procesamiento de lenguaje natural, construcción de modelos de aprendizaje automático y evaluación de su desempeño. Una vez que estas bibliotecas fueron importadas, se pudieron utilizar en el flujo de trabajo para abordar las tareas específicas del proyecto.

3.9.1.2. Carga del Dataset

A continuación, se cargó el dataset modificado, el cual fue adaptado a partir del conjunto de datos original de UCI Machine Learning. Este proceso implicó la adquisición de los datos desde su fuente, que pudo haber sido un archivo local, una base de datos o una API web. La versión modificada del dataset incluye mensajes traducidos al español y filtrados para garantizar que mantuvieran su significado en el nuevo idioma, permitiendo así una mejor evaluación del modelo en un entorno bilingüe.

Figura 20

Carga del dataset

```
# Cargar los datos en Google Colab
from google.colab import files  # Importa el módulo files desde google.colab para cargar archivos
uploaded = files.upload()  # Utiliza la función upload() para cargar archivos desde el sistema local al entorno de Google Colab
```

```
Elegir archivos  dataset.csv
• dataset.csv(text/csv) - 972372 bytes, last modified: 12/5/2024 - 100% done
Saving dataset.csv to dataset.csv
```

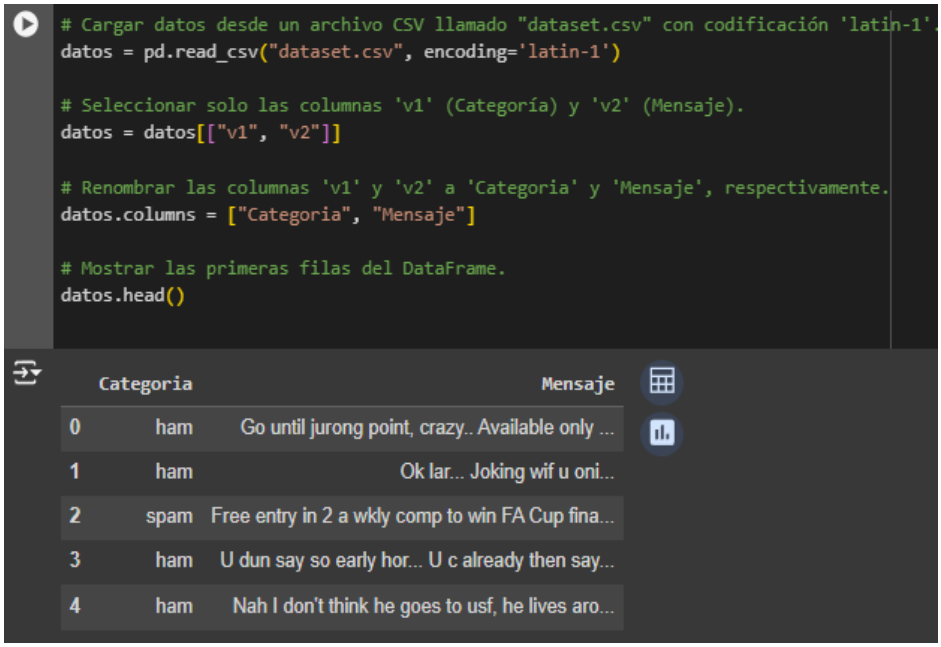
3.9.1.3. Exploración del Dataset

En el siguiente paso, se realizó una exploración en el conjunto de datos. Esta acción permitió obtener una visión general de la distribución de las categorías presentes en los datos. Fue un procedimiento útil para comprender la frecuencia con la que aparecían diferentes categorías en el conjunto de datos y esencial para entender la distribución de los datos antes de realizar análisis más avanzados o modelado predictivo.

Se cargaron los datos desde el archivo CSV llamado "dataset.csv" como se aprecia en la Figura 20, utilizando la función read_csv de Pandas. Se especificó la codificación 'latin-1' para manejar posibles caracteres especiales. Luego, se seleccionaron las columnas 'v1' (Categoría) y 'v2' (Mensaje) del conjunto de datos. Estas columnas se renombraron como 'Categoría' y 'Mensaje' respectivamente. Finalmente, se mostraron las primeras filas del DataFrame resultante mediante el método head.

Figura 21

Exploración del dataset



```
# Cargar datos desde un archivo CSV llamado "dataset.csv" con codificación 'latin-1'.
datos = pd.read_csv("dataset.csv", encoding='latin-1')

# Seleccionar solo las columnas 'v1' (Categoría) y 'v2' (Mensaje).
datos = datos[["v1", "v2"]]

# Renombrar las columnas 'v1' y 'v2' a 'Categoria' y 'Mensaje', respectivamente.
datos.columns = ["Categoria", "Mensaje"]

# Mostrar las primeras filas del DataFrame.
datos.head()
```

	Categoria	Mensaje
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

3.9.1.4. Conteo de Categorías en el Dataset

En esta etapa, se realizó el conteo de las ocurrencias de cada valor único en la columna "Categoría" del conjunto de datos modificado, utilizando el método value_counts de Pandas. Como se muestra en la Figura 22, el dataset incluye cuatro categorías: ham (mensajes legítimos en inglés), ham_es (mensajes legítimos en español), spam (mensajes de spam en inglés) y spam_es (mensajes de spam en español). Este análisis permitió verificar la correcta integración de los datos traducidos y asegurar un balance adecuado entre las distintas clases del dataset.

Figura 22

Frecuencia de ocurrencia de los datos

```
# Contar el número de ocurrencias de cada valor único en la columna 'Categoria'
datos["Categoria"].value_counts()
```

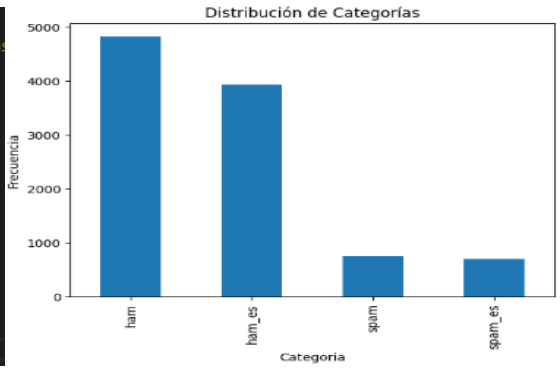
Categoria	count
ham	4825
ham_es	3939
spam	747
spam_es	698

Además, se creó un gráfico de barras para visualizar la distribución de estas categorías, empleando el método plot con el argumento kind='bar'. También se añadieron etiquetas en el eje x (xlabel) y en el eje y (ylabel) para especificar la categoría y su frecuencia. La Figura 22 ilustra esta distribución, donde la categoría ham cuenta con 4,825 mensajes, ham_es con 3,939, spam con 747 y spam_es con 698, lo que permite analizar el equilibrio del conjunto de datos.

Figura 23

Representación gráfica de la ocurrencia de los datos

```
# Cuenta las ocurrencias de cada valor único en la columna 'Categoria'
# y crea un gráfico de barras para visualizar la distribución de las categorías
datos['Categoria'].value_counts().plot(kind='bar')

# Etiqueta del eje x del gráfico
plt.xlabel('Categoria')

# Etiqueta del eje y del gráfico
plt.ylabel('Frecuencia')

# Título del gráfico
plt.title('Distribución de Categorías')

# Muestra el gráfico
plt.show()
```



3.9.2. Fase de Preprocesamiento de Datos

En esta fase, se aplicaron diversas técnicas para limpiar y estructurar los mensajes del dataset antes de ser utilizados en los modelos de detección de spam. El preprocesamiento es una etapa fundamental en el procesamiento de lenguaje natural (NLP), ya que permite reducir el ruido en los datos y mejorar el desempeño de los algoritmos.

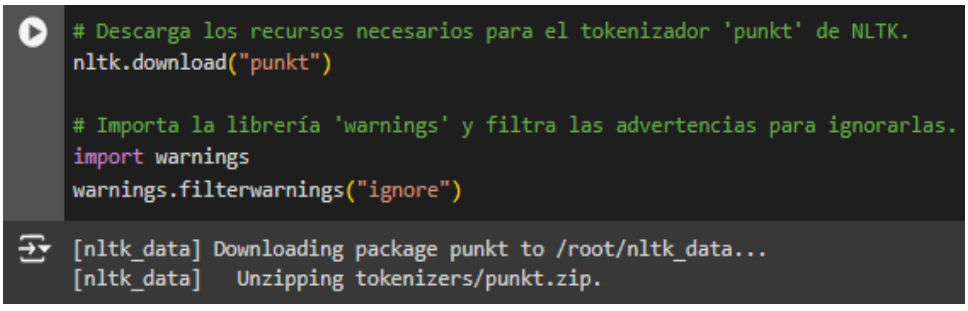
3.9.2.1. Descarga de Recursos para Tokenización

Para llevar a cabo la tokenización de los mensajes, se descargó el paquete "punkt" de la biblioteca NLTK (Natural Language Toolkit), el cual proporciona modelos preentrenados para la segmentación de texto en oraciones y palabras. La descarga se realizó mediante el comando nltk.download ("punkt"), asegurando la disponibilidad de los recursos necesarios para el procesamiento del texto.

Además, se importó la librería warnings y se configuró para ignorar advertencias mediante warnings.filterwarnings ("ignore"), con el objetivo de evitar mensajes innecesarios en la ejecución del código. La Figura 24 muestra este proceso, incluyendo la descarga y descompresión del paquete en el directorio de trabajo.

Figura 24

Descarga de las herramientas para tokenizar los datos



3.9.2.2. Tokenización del Conjunto de Datos

Para llevar a cabo la tokenización de los mensajes etiquetados como "spam" y "ham" en inglés y español, se inicializaron listas vacías para almacenar los tokens de cada categoría. Se iteró sobre los mensajes, convirtiendo el texto a minúsculas para normalizarlo y asegurar que palabras como "Casa" y "casa" sean tratadas igual. Luego, se aplicó la función nltk.mord_tokenize para segmentar el texto en tokens, los cuales se agregaron a las listas correspondientes: spam_tokens (spam en inglés), ham_tokens (ham en inglés), ham_es_tokens (ham en español) y spam_es_tokens (spam en español).

Este proceso es fundamental para el análisis de texto, ya que descompone los mensajes en unidades más pequeñas (tokens) útiles para tareas como clasificación o generación de nubes de palabras. La normalización a minúsculas mejora la precisión de los modelos de procesamiento de lenguaje natural. La Figura 25 muestra este proceso, destacando la iteración y tokenización de los mensajes.

Figura 25

Tokenización del conjunto de datos

```
# Lista de tokens para mensajes de spam
spam_tokens = []  # Inicializa una lista para almacenar los tokens de los mensajes de spam
for mensaje in datos[datos['Categoria'] == 'spam'].Mensaje:  # Itera sobre los mensajes etiquetados como 'spam' en el DataFrame
    mensaje = mensaje.lower()  # Convierte el mensaje a minúsculas
    tokens = nltk.word_tokenize(mensaje)  # Tokeniza el mensaje
    spam_tokens.extend(tokens)  # Agrega los tokens a la lista de tokens de spam

# Lista de tokens para mensajes de ham
ham_tokens = []  # Inicializa una lista para almacenar los tokens de los mensajes de ham
for mensaje in datos[datos['Categoria'] == 'ham'].Mensaje:  # Itera sobre los mensajes etiquetados como 'ham' en el DataFrame
    mensaje = mensaje.lower()  # Convierte el mensaje a minúsculas
    tokens = nltk.word_tokenize(mensaje)  # Tokeniza el mensaje
    ham_tokens.extend(tokens)  # Agrega los tokens a la lista de tokens de ham

# Lista de tokens para mensajes de ham
ham_es_tokens = []  # Inicializa una lista para almacenar los tokens de los mensajes de ham
for mensaje in datos[datos['Categoria'] == 'ham_es'].Mensaje:  # Itera sobre los mensajes etiquetados como 'ham' en el DataFrame
    mensaje = mensaje.lower()  # Convierte el mensaje a minúsculas
    tokens = nltk.word_tokenize(mensaje)  # Tokeniza el mensaje
    ham_es_tokens.extend(tokens)  # Agrega los tokens a la lista de tokens de ham

spam_es_tokens = []  # Inicializa una lista para almacenar los tokens de los mensajes de spam
for mensaje in datos[datos['Categoria'] == 'spam_es'].Mensaje:  # Itera sobre los mensajes etiquetados como 'spam' en el DataFrame
    mensaje = mensaje.lower()  # Convierte el mensaje a minúsculas
    tokens = nltk.word_tokenize(mensaje)  # Tokeniza el mensaje
    spam_es_tokens.extend(tokens)  # Agrega los tokens a la lista de tokens de spam
```

3.9.2.3. Generación de Nubes de Palabras para Spam y Ham

Para visualizar las palabras más frecuentes en los mensajes de spam y ham, tanto en inglés como en español, se generaron nubes de palabras. Primero, se convirtieron las listas de tokens en cadenas de texto utilizando join. Esto permitió unificar los tokens en un solo bloque de texto para cada categoría. Luego, se utilizó la librería wordcloud para crear las nubes de palabras, configurando el tamaño de la imagen, el color de fondo y el tamaño mínimo de la fuente. Las nubes de palabras se generaron para cuatro categorías: spam en inglés (spam_text), ham en inglés (ham_text), spam en español (spam_es_text) y ham en español (ham_es_text), como se muestra en la Figura 26.

Figura 26

Conversión de listas de tokens a cadenas

```
import matplotlib.pyplot as plt  # Importa la librería matplotlib para visualización de gráficos

# Convertir las listas de tokens a cadenas
spam_text = ' '.join(spam_tokens)  # Convierte la lista de tokens de spam a una cadena de texto
ham_text = ' '.join(ham_tokens)  # Convierte la lista de tokens de ham a una cadena de texto
spam_es_text = ' '.join(spam_es_tokens)  # Convierte la lista de tokens de spam a una cadena de texto
ham_es_text = ' '.join(ham_es_tokens)  # Convierte la lista de tokens de ham a una cadena de texto


# Crear la nube de palabras para spam
nube_palabras_spam = WordCloud(width=800, height=800,
                               background_color='white',
                               stopwords=None,
                               min_font_size=10).generate(spam_text)

# Crear la nube de palabras para ham
nube_palabras_ham = WordCloud(width=800, height=800,
                              background_color='white',
                              stopwords=None,
                              min_font_size=10).generate(ham_text)

# Crear la nube de palabras para spam en español
nube_palabras_spam_es = WordCloud(width=800, height=800,
                               background_color='white',
                               stopwords=None,
                               min_font_size=10).generate(spam_es_text)

# Crear la nube de palabras para ham en español
nube_palabras_ham_es = WordCloud(width=800, height=800,
                              background_color='white',
                              stopwords=None,
                              min_font_size=10).generate(ham_es_text)
```

Este proceso es útil para identificar visualmente las palabras más recurrentes en cada categoría, lo que ayuda a entender las características distintivas de los mensajes de spam y ham. Las nubes de palabras permiten una interpretación rápida y clara de los términos más comunes,

facilitando el análisis de patrones en los textos y proporcionando información valiosa para mejorar estrategias de filtrado y clasificación de correos electrónicos. La Figura 27 muestra la nube de palabras generada para la categoría spam, destacando las palabras más recurrentes en este tipo de mensajes.

Figura 27

Visualización de palabras con mayor repitencia en mensajes de spam

Se repitió el proceso anterior para los mensajes etiquetados como Ham, Spam_es y Ham_es. Esto facilitó la identificación de patrones lingüísticos y términos más frecuentes en los mensajes de correo electrónico legítimos y spam, como se ilustran en las Figuras 28, 29 y 30.

Figura 28

Visualización de palabras con mayor repitencia en mensajes de ham

Figura 29

Visualización de palabras con mayor repitencia en mensajes de ham en español

```
# Mostrar la nube de palabras para los mensajes de ham
plt.figure(figsize=(7, 5))  # Crea una figura con un tamaño de 8x8 pulgadas
plt.imshow(nube_palabras_ham_es, aspect='auto')  # Muestra la nube de palabras para los mensajes de ham español
plt.axis("off")  # Desactiva los ejes del gráfico
plt.title('Nube de Palabras para Mensajes de Ham en Español')  # Establece el título del gráfico
plt.show()  # Muestra el gráfico
```

Figura 30

Visualización de palabras con mayor repitencia en mensajes de spam en español

```
# Mostrar la nube de palabras para los mensajes de sham
plt.figure(figsize=(7, 5))  # Crea una figura con un tamaño de 8x8 pulgadas
plt.imshow(nube_palabras_spam_es, aspect='auto')  # Muestra la nube de palabras para los mensajes de spam español
plt.axis("off")  # Desactiva los ejes del gráfico
plt.title('Nube de Palabras para Mensajes de Spam en Español')  # Establece el título del gráfico
plt.show()  # Muestra el gráfico
```

3.9.2.4. Asignación de Etiquetas Numéricas

Para facilitar el procesamiento de los datos en los modelos de clasificación, se asignaron etiquetas numéricas a las categorías de los mensajes. Utilizando la función map, se transformaron las categorías textuales en valores numéricos: "ham" y "ham_es" se asignaron a 0 (mensajes legítimos), mientras que "spam" y "spam_es" se asignaron a 1 (mensajes no deseados). Estos valores se almacenaron en una nueva columna llamada etiqueta.

La Figura 31 muestra un fragmento del DataFrame con las nuevas etiquetas numéricas, donde se puede observar la correspondencia entre las categorías originales y los valores asignados. Este paso es crucial para preparar los datos antes de aplicar algoritmos de machine learning, ya que permite trabajar con valores numéricos en lugar de texto.

Figura 31

Asignación de etiquetas numéricas

```
# Asignar etiquetas numéricas a la columna 'Categoria' y almacenarlas en una nueva columna llamada 'etiqueta'
datos['etiqueta'] = datos['Categoria'].map({'ham': 0, 'spam': 1, 'spam_es': 1, 'ham_es': 0})

# Imprimir las primeras 15 filas del DataFrame 'datos' para verificar los cambios
print(datos.head(10000))
```

```
     Categoria                                            Mensaje  etiqueta
0          ham  Go until jurong point, crazy.. Available only ...         0
1          ham                      Ok lar... Joking wif u oni...         0
2         spam  Free entry in 2 a wkly comp to win FA Cup fina...         1
3          ham  U dun say so early hor... U c already then say...         0
4          ham  Nah I don't think he goes to usf, he lives aro...         0
...        ...                                                ...       ...
9934    ham_es    Oye, lo siento, estaba en la ducha, ¿qué pasa?         0
9935    ham_es   Carlos está abajo, pero tengo que recogerlo, ...         0
9936    ham_es           Calor total pa:-) he aplicado aceite pa.         0
9937    ham_es   Estoy atrapado en la mitad de la fila del lad...         0
9938    ham_es   ¿Has dejado tu línea de Airtel para descansar...         0

[9939 rows x 3 columns]
```

3.9.2.5. Importación de Librerías para Manipulación del Texto

En esta sección, se importaron las librerías y recursos necesarios para el procesamiento de texto. Se utilizó el módulo string para manipular cadenas de texto, mientras que la librería NLTK proporcionó herramientas clave como la lista de stopwords (palabras comunes que no aportan significado, como "el" o "y"), la función word_tokenize para dividir el texto en

palabras individuales (tokens), y el algoritmo PorterStemmer para reducir las palabras a su forma base (stemming). Estos recursos son esenciales para preparar y limpiar el texto antes de aplicar técnicas de análisis o modelado.

Figura 32

Importación de librerías para manipulación del texto

```
import string  # Importa el módulo string para manipulaciones de cadenas de texto
from nltk.corpus import stopwords  # Importa la lista de stopwords de NLTK
from nltk.tokenize import word_tokenize  # Importa la función word_tokenize de NLTK para tokenizar palabras
from nltk.stem import PorterStemmer  # Importa el algoritmo de stemming PorterStemmer de NLTK
```

3.9.2.6. Definición de Función para el Procesamiento de los Datos

Para preparar los mensajes antes de su análisis, se implementó la función Mensaje_procesado. Esta función realiza varias tareas de limpieza y normalización del texto: primero, elimina los signos de puntuación utilizando el módulo string. Luego, tokeniza el mensaje en palabras individuales con word_tokenize. Después, aplica el algoritmo PorterStemmer para reducir las palabras a su forma base (stemming) y elimina las stopwords (palabras comunes sin significado relevante) en inglés y español. Finalmente, el mensaje procesado se devuelve como una cadena de texto unificada. Este proceso es fundamental para mejorar la calidad de los datos y facilitar su uso en modelos de clasificación.

Figura 33

Definición de función para el procesamiento de los datos

```
def Mensaje_procesado(Mensaje):
    # Eliminar signos de puntuación
    Mensaje = Mensaje.translate(str.maketrans('', '', string.punctuation))

    # Tokenizar el mensaje
    tokens = word_tokenize(Mensaje)

    # Inicializar el stemmer
    stemmer = PorterStemmer()

    # Eliminar stopwords y lematizar
    Mensaje = [stemmer.stem(word.lower()) for word in tokens if word.lower() not in stopwords.words('english','spanish')]

    return " ".join(Mensaje)
```

3.9.2.7. Aplicación de la Función de Procesamiento

Para garantizar que todos los mensajes estén correctamente procesados, se aseguró que los datos en la columna Mensaje fueran de tipo string. Luego, se aplicó la función Mensaje_procesado a cada mensaje en esta columna. Esta función realiza tareas como la eliminación de signos de puntuación, tokenización, eliminación de stopwords y stemming, lo que normaliza el texto y lo prepara para su uso en modelos de clasificación.

La Figura 34 muestra un fragmento del DataFrame después de aplicar el procesamiento, donde se observa cómo los mensajes han sido transformados a un formato más limpio y estandarizado. Este paso es crucial para mejorar la calidad de los datos y asegurar que los modelos de machine learning puedan trabajar con información consistente y relevante.

Figura 34

Verificación del procesamiento de los datos

```
# Asegura que todos los datos sean de tipo string y aplica la función 'mensaje_procesado' a la columna 'Mensaje'
datos['Mensaje'] = datos['Mensaje'].astype(str).apply(Mensaje_procesado)

# Imprime las primeras 10 filas del DataFrame 'datos' para verificar los cambios
datos.head(7500)
```

	Categoria	Mensaje	etiqueta
0	ham	go jurong point crazi avail bugi n great world...	0
1	ham	ok lar joke wif u oni	0
2	spam	free entri 2 wkli comp win fa cup final tkt 21...	1
3	ham	u dun say earli hor u c alreadi say	0
4	ham	nah dont think goe usf live around though	0
...	...	...	...
7495	ham_es	buena noch voy dormir	0
7496	ham_es	de acuerdo también tomaré algo de comer mándam...	0
7497	ham_es	¿por qué pued venir aquí buscar trabajo	0
7498	ham_es	toma algo para el dolor si se muev hacia cualq...	0
7499	ham_es	jaja oh cariño deslizaré hacia tu casa despué ...	0

7500 rows × 3 columns

3.9.2.8. Creación de un Nuevo Dataset con los Datos Procesados

Una vez que los mensajes fueron procesados y normalizados, se creó un nuevo DataFrame llamado datos_procesados que contiene únicamente las columnas Mensaje y Categoría. Este DataFrame almacena los mensajes limpios y sus respectivas etiquetas numéricas, lo que facilita su uso en tareas de modelado y clasificación.

La Figura 35 muestra un fragmento de este DataFrame, donde se puede observar cómo los mensajes han sido transformados a un formato más limpio y estandarizado, listos para ser utilizados en algoritmos de machine learning. Este paso es fundamental para asegurar que los datos estén en un formato adecuado para su análisis y predicción.

Figura 35

Creación de un nuevo dataset con los datos procesados

```
# Crear un DataFrame con los datos procesados
datos_procesados = pd.DataFrame({'Mensaje': datos['Mensaje'], 'Categoria': datos['etiqueta']})

# Imprimir las primeras 10 filas del DataFrame 'datos_procesados' para verificar los cambios
datos_procesados.head(7000)
```

	Mensaje	Categoria
0	go jurong point crazi avail bugi n great world...	0
1	ok lar joke wif u oni	0
2	free entri 2 wkli comp win fa cup final tkt 21...	1
3	u dun say earli hor u c alreadi say	0
4	nah dont think goe usf live around though	0
...	...	...
6995	gracia	0
6996	mándame un mensaj cuando llegu rajini	0
6997	¡sí ¿cuándo va tomar tu clase práctica yo comi...	0
6998	eso está bien porqu necesito droga	0
6999	estúpido es posibl	0

7000 rows × 2 columns

3.9.3. Fase de Vectorización

En esta fase, los mensajes de texto procesados se transforman en representaciones numéricas para que puedan ser utilizados por los algoritmos de machine learning. La

vectorización convierte el texto en estructuras numéricas, como vectores, que capturan la frecuencia de las palabras o su importancia relativa en el conjunto de datos. Este proceso es esencial porque los modelos de machine learning requieren datos numéricos para realizar cálculos y predicciones. En esta sección, se utilizaron técnicas como la vectorización basada en conteo y TF-IDF (Term Frequency-Inverse Document Frequency) para representar los mensajes de manera efectiva.

3.9.3.1. Conteo de Palabras en el Dataset

Para entender la distribución de las palabras en el dataset, se utilizó la clase Counter de la librería collections. Este proceso contabiliza cuántas veces aparece cada palabra en la columna Mensaje. Se recorrieron todos los mensajes y se dividieron en palabras individuales, incrementando el conteo de cada palabra en el diccionario total_counts. Al final, se obtuvo un total de 16,487 palabras únicas en el dataset, como se muestra en la Figura 36. Este análisis es útil para identificar la frecuencia de las palabras y preparar el texto para su vectorización, lo que facilita la creación de modelos de machine learning más efectivos.

Figura 36

Numeración del total de palabras en el dataset

```
from collections import Counter  # Importar la clase Counter

# Contar cuántas veces una palabra aparece en el dataset dentro de la columna 'Mensaje'
total_counts = Counter()
for mensaje in datos['Mensaje']:
    for palabra in mensaje.split():
        total_counts[palabra] += 1

print("Total de palabras en el dataset:", len(total_counts))
```

```
Total de palabras en el dataset: 16487
```

3.9.3.2. Ordenación y Mapeo de Palabras

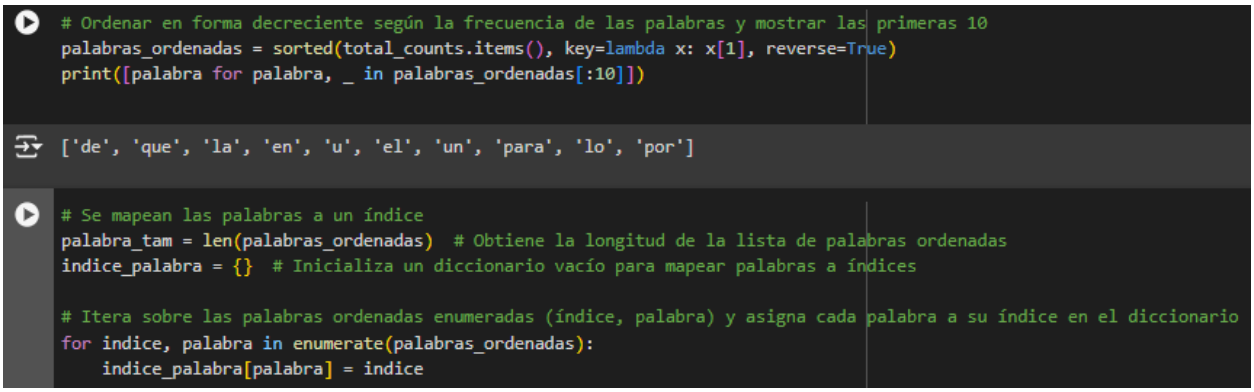
Una vez obtenido el conteo de palabras, se ordenaron en forma decreciente según su frecuencia en el dataset. Las 10 palabras más frecuentes fueron: ['de', 'que', 'la', 'en', 'u', 'el', 'un',

'para', 'lo', 'por']. Este ordenamiento permite identificar las palabras más comunes, lo que es útil para entender la estructura del texto y priorizar términos relevantes en el análisis.

Posteriormente, se mapeó cada palabra a un índice único utilizando un diccionario (indice_palabra). Este mapeo es esencial para convertir las palabras en representaciones numéricas. La Figura 37 ilustra este proceso de ordenación y mapeo, destacando cómo se asignaron índices a las palabras según su frecuencia.

Figura 37

Mapeo de palabras a índices



3.9.3.3. Conversión de Mensajes a Vectores

Para convertir los mensajes de texto en representaciones numéricas, se implementó la función mensaje_a_vector. Esta función toma un mensaje y lo transforma en un vector de tamaño igual al vocabulario del dataset. Cada posición del vector corresponde a una palabra específica, y su valor indica la frecuencia de esa palabra en el mensaje. Si la palabra está en el vocabulario, se incrementa su conteo en el vector correspondiente. Luego, se creó una matriz de ceros para almacenar los vectores de todos los mensajes, y se iteró sobre cada mensaje para aplicar la función mensaje_a_vector. Este proceso, ilustrado en la Figura 38, permite representar los mensajes en un formato numérico adecuado para su uso en modelos de machine learning.

Figura 38

Conversión de los mensajes de textos a vectores

```
# Se vuelca texto al vector
def mensaje_a_vector(texto):
    vector_mensaje = np.zeros(palabra_tam)  # Crea un vector de ceros del tamaño del vocabulario
    for palabra in texto.split(" "):  # Itera sobre cada palabra en el texto
        indice = indice_palabra.get(palabra)  # Obtiene el índice de la palabra en el vocabulario
        if indice is not None:  # Verifica si la palabra está en el vocabulario
            vector_mensaje[indice] += 1  # Incrementa el conteo de la palabra en el vector
    return np.array(vector_mensaje)  # Convierte el vector a un array numpy

vector = np.zeros((len(datos), palabra_tam), dtype=np.int_)  # Crea una matriz de ceros para almacenar los vectores de palabras
for i, (_, mensaje_) in enumerate(datos['Mensaje'].items()):  # Itera sobre cada mensaje en el DataFrame
    vector[i] = mensaje_a_vector(mensaje_)  # Convierte el mensaje a un vector y lo almacena en la matriz
```

3.9.3.4. Verificación y Dimensionamiento del Vector:

Una vez completada la vectorización de los mensajes, se obtuvo una matriz donde cada fila representa un mensaje y cada columna corresponde a una palabra del vocabulario. La forma de esta matriz, mostrada en la Figura 39, es (10209, 16487), lo que indica que hay 10,209 mensajes y 16,487 palabras únicas en el vocabulario. Esta estructura permite representar cada mensaje como un vector numérico, lo que es esencial para el entrenamiento de modelos de machine learning, ya que proporciona una entrada adecuada para algoritmos que requieren datos numéricos.

Figura 39

Verificación de los valores del vector creado

```
# Devuelve la forma del array 'vector', que contiene los vectores de palabras para cada mensaje en 'datos'
vector.shape
```

```
(10209, 16487)
```

3.9.3.5. Conversión de los Datos a Vectores mediante TfidfVectorizer

Para mejorar la representación de los mensajes, se utilizó la técnica TF-IDF (Term Frequency-Inverse Document Frequency) a través de la clase TfidfVectorizer de la librería sklearn. Esta técnica no solo considera la frecuencia de las palabras en un mensaje (TF), sino

también su importancia relativa en todo el dataset (IDF). Esto permite reducir el peso de palabras comunes y destacar aquellas más relevantes. La matriz resultante, mostrada en la Figura 39, tiene una forma de (10209, 15866), lo que indica que hay 10,209 mensajes y 15,866 características (palabras únicas). Esta representación es más eficaz para modelos de machine learning, ya que captura mejor la importancia de las palabras en el contexto del dataset.

Figura 40

Conversión de los datos a vectores mediante TfidfVectorizer

```
# Importar la clase TfidfVectorizer de sklearn para convertir datos de texto en vectores utilizando TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer

# Utilizar TfidfVectorizer para transformar los datos de texto en vectores TF-IDF y almacenar el resultado en 'vectors'
vectors = TfidfVectorizer().fit_transform(datos['Mensaje'])

# Devolver la forma de la matriz de vectores 'vectors', que representa la cantidad de documentos (filas) y características (columnas)
vectors.shape
```

```
(10209, 15866)
```

3.9.4. Fase de Entrenamiento y Valoración de Modelos

En esta fase, se entrenaron y evaluaron tres algoritmos de aprendizaje automático: Naive Bayes, K-Nearest Neighbors (KNN) y Decision Tree, utilizando un conjunto de datos de correos electrónicos etiquetados como spam o no spam. El proceso de entrenamiento involucró la división del dataset en conjuntos de entrenamiento (80%) y prueba (20%), asegurando la validación cruzada para una evaluación precisa. Esta división se realizó siguiendo criterios empíricos y estadísticos que han demostrado que utilizar entre el 70% y 80% de los datos para entrenamiento y el 20-30% para prueba permite optimizar el aprendizaje del modelo sin comprometer su capacidad de generalización (Gholamy et al., 2018). Cada modelo fue ajustado y optimizado para maximizar su rendimiento. Posteriormente, se evaluaron las métricas de rendimiento como precisión, recall, F1-score y la tasa de falsos positivos y negativos para determinar el modelo más eficaz para la detección de spam.

3.9.4.1. División del Conjunto de Datos en Entrenamiento y Prueba

Para evaluar el rendimiento de los modelos de machine learning, se dividió el dataset en dos conjuntos: entrenamiento y prueba. Utilizando la función train_test_split de la librería scikit-learn, se asignó el 80% de los datos para entrenamiento (X_entrenamiento, y_entrenamiento) y el 20% para pruebas (X_prueba, y_prueba). La división se realizó de manera aleatoria, pero con una semilla (random_state=100) para garantizar la reproducibilidad de los resultados, lo que significaba que produciría la misma división cada vez que se ejecutara el código. Este paso, ilustrado en la Figura 41, es crucial para entrenar los modelos con una parte de los datos y validar su rendimiento con otra parte no vista durante el entrenamiento.

Figura 41

División del conjunto de datos en set de entrenamiento y prueba

```
# Importar la función train_test_split de scikit-learn
from sklearn.model_selection import train_test_split

# Dividir el conjunto de datos en set de entrenamiento y prueba
X_entrenamiento, X_prueba, y_entrenamiento, y_prueba = train_test_split(vectors, datos['etiqueta'], test_size=0.20, random_state=100)
```

3.9.4.2. Métricas de Evaluación de Rendimiento de los algoritmos

Para evaluar el rendimiento de los algoritmos de detección de spam, se utilizaron varias métricas que proporcionaron diferentes perspectivas sobre la eficacia del modelo. Estas métricas fueron cruciales para entender cómo se comportaba el algoritmo en términos de precisión, capacidad de detección y balance entre errores. A continuación, se describen las métricas de exactitud, precisión, recall y F1 Score, explicando su relevancia y las fórmulas para su cálculo.

- **Exactitud**

La exactitud (accuracy) es una métrica crucial que mide la proporción de predicciones correctas hechas por un algoritmo en comparación con el total de predicciones. Es una medida

simple y directa que ofrece una visión general del rendimiento del modelo (Vujović, 2021). La fórmula para calcular la exactitud es:

$$\textbf{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN} \qquad (8)$$

Donde:

- VP (Verdaderos Positivos) son los verdaderos positivos, es decir, los casos correctamente identificados como positivos.
- TN (Verdaderos Negatives) son los verdaderos negativos, es decir, los casos correctamente identificados como negativos.
- FP (Falsos Positivos) son los falsos positivos, es decir, los casos incorrectamente identificados como positivos.
- FN (Falsos Negativos) son los falsos negativos, es decir, los casos incorrectamente identificados como negativos.

- **Precisión**

La precisión es una métrica que evalúa la calidad de las predicciones positivas del modelo, indicando la proporción de verdaderos positivos en relación con el total de predicciones positivas. Una alta precisión implica que el modelo tiene pocos falsos positivos, lo cual es esencial en aplicaciones donde los errores positivos son costosos (Vujović, 2021). La precisión se calcula con la siguiente fórmula:

$$\textbf{Precisión} = \frac{VP}{VP + FP} \qquad (9)$$

Donde:

- VP (Verdaderos Positivos) son los verdaderos positivos, es decir, los casos correctamente identificados como positivos.
- FP (Falsos Positivos) son los falsos positivos, es decir, los casos incorrectamente identificados como positivos.

- **Recall**

El recall, también conocido como sensibilidad o tasa de verdaderos positivos, mide la capacidad del modelo para identificar correctamente todos los ejemplos positivos. Esta métrica se enfoca en la proporción de verdaderos positivos en relación con el total de ejemplos que realmente son positivos (Vujović, 2021). El recall se calcula con la siguiente fórmula:

$$\textbf{Recall} = \frac{VP}{VP + FN} \qquad (10)$$

Donde:

- VP (Verdaderos Positivos) son los verdaderos positivos, es decir, los casos correctamente identificados como positivos.
- FN (Falsos Negativos) son los falsos negativos, es decir, los casos que son positivos pero que el modelo identificó incorrectamente como negativos.

- **F1 Score**

El F1 Score es una métrica que equilibra la precisión y el recall al combinarlos en una sola medida. Es especialmente útil cuando se necesita un balance entre la capacidad del modelo para identificar correctamente los ejemplos positivos y su precisión en esas predicciones (Vujović, 2021). El F1 Score se calcula como la media armónica de la precisión y el recall:

$$\textbf{F1 Score} = 2 \times \frac{Precisión \times Recall}{Precisión + Recall} \qquad (11)$$

Donde:

- La precisión se calcula como: $\mathbf{Precisión} = \frac{VP}{VP+FP}$
- El recall se calcula como: $\mathbf{Recall} = \frac{VP}{VP+FN}$

- **AUC-ROC**

El AUC-ROC (Área Bajo la Curva - Curva Característica Operativa del Receptor) es una métrica que evalúa el rendimiento de un modelo de clasificación en todos los umbrales de probabilidad posibles. La curva ROC se traza con la tasa de verdaderos positivos (TPR) en el eje Y y la tasa de falsos positivos (FPR) en el eje X. El AUC representa el área total bajo esta curva, proporcionando una medida única de rendimiento del modelo (Vujović, 2021).

$$\mathbf{AUC - ROC} = \frac{Recall + Especifiicidad}{2} \qquad (\ 12\ )$$

Donde:

- Recall mide la proporción de casos positivos correctamente identificados por el modelo.
- Especificidad mide la proporción de casos negativos correctamente identificados por el modelo.

3.9.4.3. Optimización de Hiperparámetros para el Modelo KNN

Para mejorar el rendimiento del clasificador K-Nearest Neighbors (KNN), se realizó una búsqueda en cuadrícula (GridSearchCV) sobre el hiperparámetro n_neighbors, probando valores como 5, 10, 20, 50, 60, 70, 80, 90 y 100. Este proceso incluyó validación cruzada de 5 subconjuntos (cv=5), dividiendo los datos de entrenamiento en 5 partes, entrenando el modelo en 4 de ellas y validándolo en la quinta, repitiendo este proceso para cada subconjunto. Esto permitió una evaluación más robusta y confiable del rendimiento del modelo.

Los resultados mostraron que el valor óptimo para n_neighbors fue 50, con una puntuación media de validación de 0.9409. Otros valores, como 5 y 10, obtuvieron puntuaciones más bajas (0.8898 y 0.8671, respectivamente), mientras que 20 y 60 alcanzaron puntuaciones cercanas al óptimo (0.9313 y 0.9392). Sin embargo, a partir de n_neighbors = 60, se observó una ligera disminución en el rendimiento, con puntuaciones de 0.9388 para 70, 0.9377 para 80, 0.9359 para 90 y 0.9354 para 100.

Esta tendencia sugiere que un aumento excesivo en el número de vecinos genera un efecto de suavización que perjudica la capacidad del modelo para capturar patrones más específicos, lo que puede llevar a una clasificación menos precisa. La Figura 42 ilustra esta variación en el rendimiento, confirmando que valores demasiado altos de n_neighbors pueden reducir la capacidad del modelo para generalizar correctamente en datos no vistos.

Figura 42

Hiperparametrización de los valores para el algoritmo KNN

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Definir los parámetros a ajustar y sus valores
parametros = {'n_neighbors': [5, 10, 20, 50, 60, 70, 80, 90, 100]}  # Definir una lista de valores para el hiperparámetro n_neighbors

# Inicializar el modelo de clasificador KNeighbors
modelo = KNeighborsClassifier()  # Inicializar un clasificador KNeighbors sin ningún hiperparámetro especificado

# Realizar la búsqueda en cuadrícula
grid_search = GridSearchCV(modelo, parametros, cv=5)  # Inicializar un objeto GridSearchCV con el modelo y parámetros definidos
grid_search.fit(X_entrenamiento, y_entrenamiento)  # Ajustar el modelo utilizando búsqueda en cuadrícula con los datos de entrenamiento

# Obtener los resultados de la búsqueda en cuadrícula
resultados = grid_search.cv_results_

# Extraer las puntuaciones de validación media y los valores de los parámetros
puntuaciones_medias = resultados['mean_test_score']
valores_parametros = parametros['n_neighbors']

# Obtener los mejores hiperparámetros encontrados durante la búsqueda en cuadrícula
mejores_hiperparametros = grid_search.best_params_

# Imprimir los mejores hiperparámetros encontrados
print("Mejores hiperparámetros encontrados:", mejores_hiperparametros)  # Imprimir los mejores hiperparámetros

print("Puntuaciones: ")
print(puntuaciones_medias)
```

```
Mejores hiperparámetros encontrados: {'n_neighbors': 50}
Puntuaciones:
[0.88980071 0.86714812 0.93130926 0.94098268 0.93926835 0.93877845
 0.93767641 0.9359626  0.93547256]
```

3.9.4.4. Entrenamiento del Modelo KNN y Predicciones en los Datos de Prueba

Una vez optimizado el modelo KNN con el mejor valor de n_neighbors (50), se procedió a entrenarlo con los datos de entrenamiento y a evaluar su rendimiento en los conjuntos de prueba y entrenamiento. Además, se generó un informe de clasificación que muestra métricas como precisión, recall y F1-score para ambas clases.

Para visualizar el desempeño del modelo, se generaron matrices de confusión tanto para el conjunto de entrenamiento como para el de prueba, como se muestra en la Figura 43. Estas matrices permiten analizar los aciertos y errores del modelo en la clasificación de cada categoría, proporcionando una visión detallada de su capacidad para distinguir entre spam y ham.

Figura 43

Entrenamiento del algoritmo KNN y predicciones en los datos de prueba.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Inicializar el modelo KNN
clasificador_knn = KNeighborsClassifier(n_neighbors=50)

# Entrenar el modelo KNN con los datos de entrenamiento
clasificador_knn.fit(X_entrenamiento, y_entrenamiento)

# Hacer predicciones en el conjunto de prueba utilizando el modelo entrenado
predicciones_knn = clasificador_knn.predict(X_prueba)
predicciones_knn_ent = clasificador_knn.predict(X_entrenamiento)

# Imprimir el informe de clasificación y la exactitud
print('KNN Pruebas')
print(classification_report(y_prueba, predicciones_knn))
print()
print('Exactitud:', accuracy_score(y_prueba, predicciones_knn))

# Imprimir el informe de clasificación y la exactitud
print('KNN Entrenamiento')
print(classification_report(y_entrenamiento, predicciones_knn_ent))
print()
print('Exactitud:', accuracy_score(y_entrenamiento, predicciones_knn_ent))

# Calcular y visualizar la matriz de confusión
cm = confusion_matrix(y_prueba, predicciones_knn)
ce = confusion_matrix(y_entrenamiento, predicciones_knn_ent)

plt.figure(figsize=(5, 5))
sns.heatmap(ce, annot=True, linewidths=0.5, linecolor="red", fmt=".0f")
plt.xlabel("y_predicción")
plt.ylabel("y_real")
plt.title('Matriz de Confusión Entrenamiento')
plt.show()

plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, linewidths=0.5, linecolor="red", fmt=".0f")
plt.xlabel("y_predicción")
plt.ylabel("y_real")
plt.title('Matriz de Confusión Pruebas')
plt.show()
```

3.9.4.5. Estimación del Rendimiento del Modelo KNN

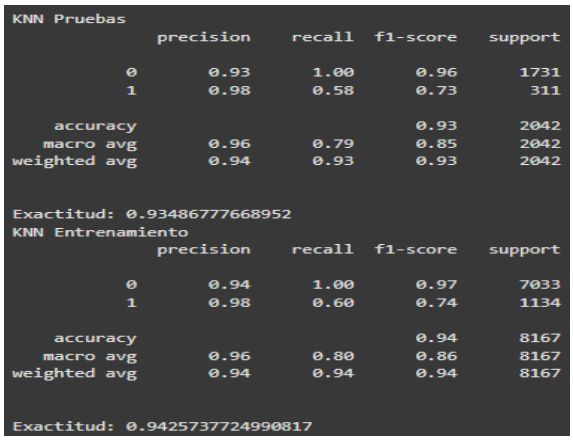
Se imprimió un informe de clasificación y la exactitud del modelo tanto para el conjunto de prueba como para el conjunto de entrenamiento, lo que proporcionó una visión detallada del rendimiento del modelo en términos de precisión, recall, f1-score y support para cada clase.

El modelo KNN, con el hiperparámetro óptimo n_neighbors=50, mostró un rendimiento sólido tanto en el conjunto de prueba como en el de entrenamiento. En el conjunto de prueba, el modelo alcanzó una exactitud (accuracy) de 0.9349, con una precisión y recall altos para la clase 0 (ham), indicando que la mayoría de los mensajes legítimos fueron clasificados correctamente. Sin embargo, para la clase 1 (spam), el recall fue más bajo (0.58), lo que sugiere que algunos mensajes de spam no fueron detectados.

En el conjunto de entrenamiento, el modelo logró una exactitud de 0.9426, con un comportamiento similar: alta precisión y recall para la clase 0, pero un recall más bajo para la clase 1. Esto indica que, aunque el modelo es efectivo para identificar mensajes legítimos, tiene dificultades para detectar todos los mensajes de spam. Estos resultados, presentados en la Figura 44, resaltan la importancia de ajustar el modelo para mejorar la detección de spam sin comprometer la precisión general.

Figura 44

Resultados obtenidos de las predicciones del modelo KNN en el conjunto de datos



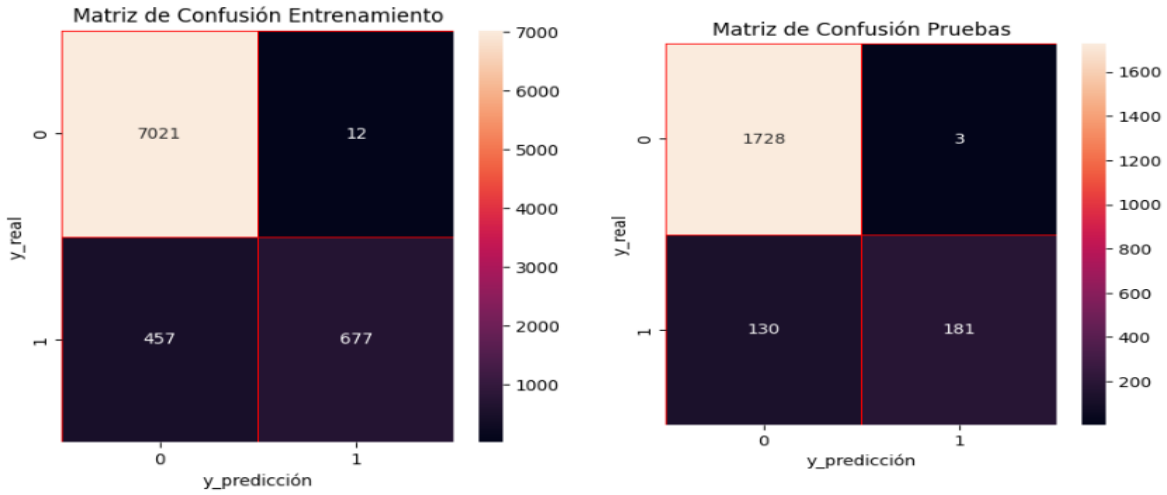
```
KNN Pruebas
              precision    recall  f1-score   support

           0       0.93      1.00      0.96      1731
           1       0.98      0.58      0.73       311

    accuracy                           0.93      2042
   macro avg       0.96      0.79      0.85      2042
weighted avg       0.94      0.93      0.93      2042


Exactitud: 0.93486777668952
KNN Entrenamiento
              precision    recall  f1-score   support

           0       0.94      1.00      0.97      7033
           1       0.98      0.60      0.74      1134

    accuracy                           0.94      8167
   macro avg       0.96      0.80      0.86      8167
weighted avg       0.94      0.94      0.94      8167


Exactitud: 0.9425737724990817
```

3.9.4.6. Matrices de Confusión del Algoritmo KNN en el Entrenamiento

La visualización de las matrices de confusión tanto para el conjunto de prueba como para el conjunto de entrenamiento mostró visualmente la calidad de las predicciones del modelo al comparar las etiquetas predichas con las etiquetas reales. Cada celda de la matriz indicó el número de instancias clasificadas correcta o incorrectamente para cada clase. La diagonal principal representó las predicciones correctas, mientras que las demás celdas indicaron errores de clasificación.

Figura 45

Matrices de confusión de los datos de entrenamiento y prueba del modelo KNN.



La Figura 45 mostró las matrices de confusión correspondientes al modelo K-Nearest Neighbors (KNN) aplicado a los conjuntos de datos de entrenamiento y prueba. Estas matrices fueron útiles para evaluar el rendimiento del modelo, permitiendo visualizar la cantidad de predicciones correctas e incorrectas que realizó el algoritmo.

Matriz de Confusión del Conjunto de Entrenamiento (Izquierda):

- **Clase 0 (No spam):** de las 7,033 instancias reales de la clase 0, el modelo clasificó correctamente 7,021 como clase 0, pero se equivocó en 12, etiquetándolas incorrectamente como clase 1

- **Clase 1 (Spam)**: de las 1,134 instancias reales de la clase 1, el modelo clasificó correctamente 677 como clase 1, pero clasificó incorrectamente 457 instancias como clase 0.

Matriz de Confusión del Conjunto de Prueba (Derecha):

- **Clase 0 (No spam):** de las 1,731 instancias reales de la clase 0, el modelo clasificó correctamente 1,728, mientras que 3 instancias fueron clasificadas incorrectamente como clase 1.
- **Clase 1 (Spam):** de las 311 instancias reales de la clase 1, 180 fueron clasificadas correctamente como clase 1, pero 130 fueron clasificadas erróneamente como clase 0.

3.9.4.7. Curva ROC y AUC del Modelo KNN

Se calculó y graficó la curva ROC (Receiver Operating Characteristic) para evaluar el desempeño del modelo de clasificación KNN en el conjunto de prueba. Para ello, primero se obtuvieron las probabilidades de pertenencia a la clase positiva ('spam') mediante el método predict_proba. Luego, se calcularon la tasa de falsos positivos (FPR) y la tasa de verdaderos positivos (TPR) utilizando la función roc_curve, tomando como parámetros las etiquetas reales (y_prueba) y las probabilidades predichas.

Finalmente, se determinó el área bajo la curva ROC (AUC) con la función roc_auc_score, proporcionando una métrica cuantitativa del rendimiento del modelo. La Figura 46 muestra la implementación de este proceso, donde se calculan las métricas mencionadas y se genera la gráfica de la curva ROC.

Figura 46

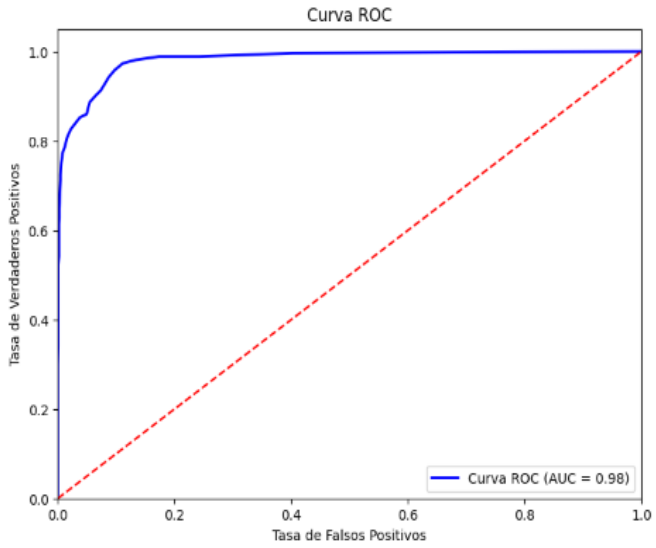
Cálculo de la curva ROC para la evaluación del desempeño de modelo KNN

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

# Obtener las probabilidades de las clases positivas
probs = clasificador_knn.predict_proba(X_prueba)[:, 1]

# Calcular la tasa de verdaderos positivos y la tasa de falsos positivos
fpr, tpr, _ = roc_curve(y_prueba, probs)

# Calcular el área bajo la curva ROC (AUC)
roc_auc = roc_auc_score(y_prueba, probs)

# Graficar la curva ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='Curva ROC (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()
```

Se graficó la curva ROC utilizando matplotlib.pyplot, Esta curva muestra la relación entre la **tasa de verdaderos positivos (TPR)** y la **tasa de falsos positivos (FPR)** a diferentes umbrales de clasificación.

Figura 47

Área bajo la curva ROC (AUC) del rendimiento del modelo KNN



La Figura 47 muestra la curva ROC (Receiver Operating Characteristic) para el modelo KNN (K-Nearest Neighbors). La curva ROC fue una herramienta visual que se utilizó para evaluar el rendimiento de un modelo de clasificación binaria, en este caso, para la detección de spam.

- **Eje X (Tasa de Falsos Positivos - FPR):** Representó la proporción de instancias negativas que fueron incorrectamente clasificadas como positivas. Un FPR bajo indicó que el modelo hizo un buen trabajo al no clasificar incorrectamente las instancias negativas.
- **Eje Y (Tasa de Verdaderos Positivos - TPR):** Representó la proporción de instancias positivas que fueron correctamente identificadas por el modelo. Un TPR alto indicó que el modelo hizo un buen trabajo al identificar correctamente las instancias positivas.
- **Línea Roja Discontinua:** Esta línea diagonal roja representó el comportamiento de un clasificador que hizo predicciones aleatorias. Cualquier modelo que tuviera su curva ROC por encima de esta línea se consideró mejor que un clasificador aleatorio.
- **Curva Azul:** Esta curva representó el rendimiento del modelo KNN. A medida que la curva se acercó al punto superior izquierdo (donde el TPR fue 1 y el FPR fue 0), mejor fue el rendimiento del modelo. Una curva ROC que se acercó al borde superior izquierdo indicó que el modelo tuvo una alta capacidad de distinguir entre clases positivas y negativas.
- **Área Bajo la Curva (AUC):** El AUC fue un valor numérico que resumió el rendimiento de la curva ROC. En este caso, el AUC fue 0.98, lo que indicó que el modelo tuvo un rendimiento excelente, ya que se acercó al valor máximo de 1.0. Un

AUC cercano a 1.0 significó que el modelo tuvo una alta capacidad para separar las clases.

3.9.4.8. Optimización de Hiperparámetros para el Modelo Naive Bayes.

Se utilizó GridSearchCV para optimizar el clasificador Naive Bayes, ajustando el hiperparámetro alpha, que controla el suavizado de Laplace. Se probaron valores de 0.1 a 1.0 con validación cruzada (cv=5), obteniendo el mejor resultado con alpha = 0.2 (puntuación media de 0.9829). Valores cercanos, como 0.1 (0.9821) y 0.3 (0.9809), también fueron competitivos, pero a partir de alpha = 0.5 el rendimiento disminuyó, con puntuaciones de 0.9762 (0.5), 0.9682 (0.7) y 0.9540 (1.0).

Este descenso sugiere que alphas mayores a 0.5 generan un suavizado excesivo, afectando la discriminación entre palabras relevantes y no relevantes para la clasificación del spam. La Figura 48 muestra cómo este efecto perjudica la efectividad del clasificador.

Figura 48

Hiperparametrización de los valores para el algoritmo Naive Bayes

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV

# Definir los parámetros a ajustar y sus valores para el clasificador Naive Bayes
parametros_nb = {'alpha': [0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 1.0]}  # Definir una lista de valores para el hiperparámetro alpha

# Inicializar el modelo de clasificador Naive Bayes
modelo_nb = MultinomialNB()  # Inicializar un clasificador Naive Bayes sin ningún hiperparámetro especificado

# Realizar la búsqueda en cuadrícula para el clasificador Naive Bayes
grid_search_nb = GridSearchCV(modelo_nb, parametros_nb, cv=5)  # Inicializar un objeto GridSearchCV para el clasificador Naive Bayes
grid_search_nb.fit(X_entrenamiento, y_entrenamiento)  # Ajustar el modelo de clasificador Naive Bayes utilizando búsqueda en cuadrícula con los datos de entrenamiento

# Obtener los resultados de la búsqueda en cuadrícula para el clasificador Naive Bayes
resultados_nb = grid_search_nb.cv_results_

# Extraer las puntuaciones de validación media y los valores de los parámetros para el clasificador Naive Bayes
puntuaciones_medias_nb = resultados_nb['mean_test_score']
valores_parametros_nb = parametros_nb['alpha']


# Obtener los mejores hiperparámetros encontrados para el clasificador Naive Bayes
mejores_hiperparametros_nb = grid_search_nb.best_params_

# Imprimir los mejores hiperparámetros encontrados para el clasificador Naive Bayes
print("Mejores hiperparámetros para Naive Bayes encontrados:", mejores_hiperparametros_nb)  # Imprimir los mejores hiperparámetros para el clasificador Naive Bayes
print(puntuaciones_medias_nb)
```

```
Mejores hiperparámetros para Naive Bayes encontrados: {'alpha': 0.2}
[0.98212307 0.98298039 0.98089908 0.97918535 0.97624657 0.96816516
 0.95408425]
```

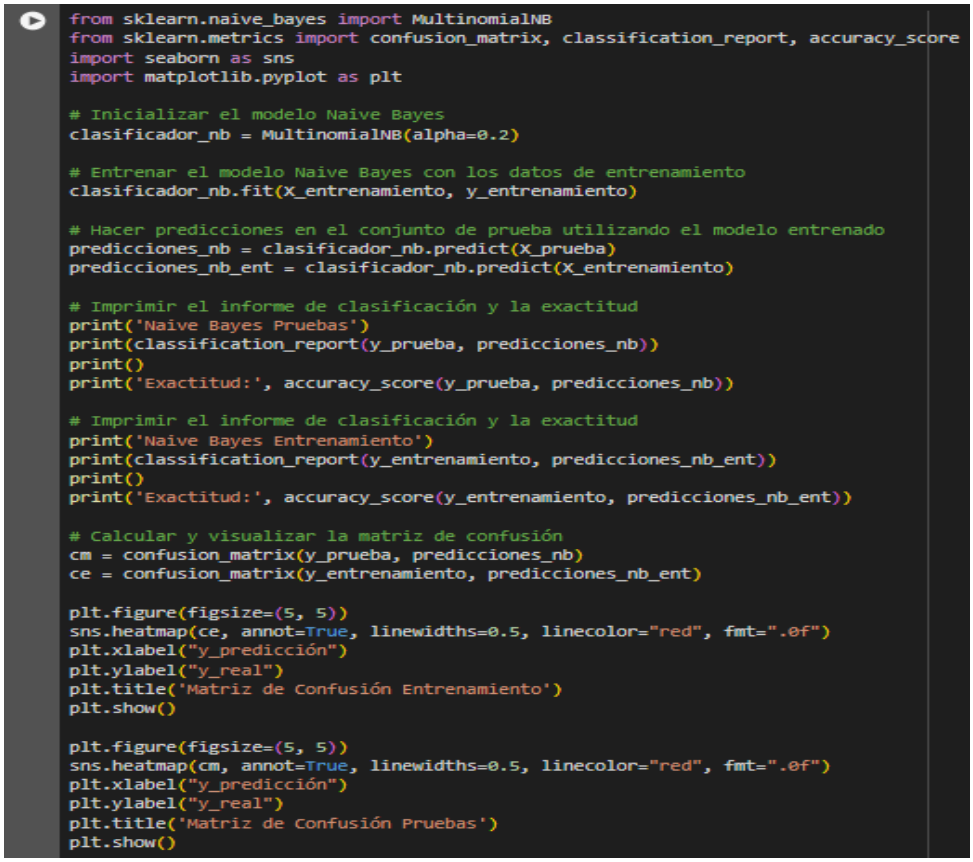
3.9.4.9. Entrenamiento del Modelo Naive Bayes y Predicciones en los Datos de Prueba

Una vez optimizado el modelo Naive Bayes con el mejor valor de alpha (0.2), se procedió a entrenarlo con los datos de entrenamiento y a evaluar su rendimiento en los conjuntos de prueba y entrenamiento. Además, se generó un informe de clasificación que muestra métricas como precisión, recall y F1-score para ambas clases.

Para visualizar el desempeño del modelo, se generaron matrices de confusión tanto para el conjunto de entrenamiento como para el de prueba, como se muestra en la Figura 49. Estas matrices permiten analizar los aciertos y errores del modelo en la clasificación de cada categoría, proporcionando una visión detallada de su capacidad para distinguir entre spam y ham.

Figura 49

Entrenamiento del algoritmo Naive Bayes y predicciones en los datos de prueba



```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Inicializar el modelo Naive Bayes
clasificador_nb = MultinomialNB(alpha=0.2)

# Entrenar el modelo Naive Bayes con los datos de entrenamiento
clasificador_nb.fit(X_entrenamiento, y_entrenamiento)

# Hacer predicciones en el conjunto de prueba utilizando el modelo entrenado
predicciones_nb = clasificador_nb.predict(X_prueba)
predicciones_nb_ent = clasificador_nb.predict(X_entrenamiento)

# Imprimir el informe de clasificación y la exactitud
print('Naive Bayes Pruebas')
print(classification_report(y_prueba, predicciones_nb))
print()
print('Exactitud:', accuracy_score(y_prueba, predicciones_nb))

# Imprimir el informe de clasificación y la exactitud
print('Naive Bayes Entrenamiento')
print(classification_report(y_entrenamiento, predicciones_nb_ent))
print()
print('Exactitud:', accuracy_score(y_entrenamiento, predicciones_nb_ent))

# Calcular y visualizar la matriz de confusión
cm = confusion_matrix(y_prueba, predicciones_nb)
ce = confusion_matrix(y_entrenamiento, predicciones_nb_ent)

plt.figure(figsize=(5, 5))
sns.heatmap(ce, annot=True, linewidths=0.5, linecolor="red", fmt=".0f")
plt.xlabel("y_predicción")
plt.ylabel("y_real")
plt.title('Matriz de Confusión Entrenamiento')
plt.show()

plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, linewidths=0.5, linecolor="red", fmt=".0f")
plt.xlabel("y_predicción")
plt.ylabel("y_real")
plt.title('Matriz de Confusión Pruebas')
plt.show()
```

3.9.4.10. Estimación del Rendimiento del Algoritmo Naive Bayes

Se generaron informes de clasificación detallados que incluyeron métricas como precisión, recall, f1-score y soporte, además de calcular la exactitud del modelo en los conjuntos de entrenamiento y prueba.

El modelo Naive Bayes, optimizado con un valor de alpha de 0.2, demostró un excelente rendimiento en la clasificación de mensajes. En el conjunto de prueba, el modelo alcanzó una exactitud (accuracy) de 0.9843, con una precisión, recall y F1-score cercanos al 99% para la clase ham (mensajes legítimos) y alrededor del 96% para la clase spam. Esto indica que el modelo es altamente efectivo para distinguir entre ambos tipos de mensajes.

En el conjunto de entrenamiento, el modelo mostró un rendimiento aún más sólido, con una exactitud de 0.9953 y métricas de precisión, recall y F1-score cercanas al 100% para ambas clases. Estos resultados, presentados en la Figura 50, confirman que el modelo no está sobreajustado y generaliza bien en datos no vistos. La alta precisión y recall en ambas clases reflejan la capacidad del modelo para identificar correctamente tanto los mensajes legítimos como los no deseados.

Figura 50

Resultados obtenidos de las predicciones en el conjunto de datos del modelo Naive Bayes

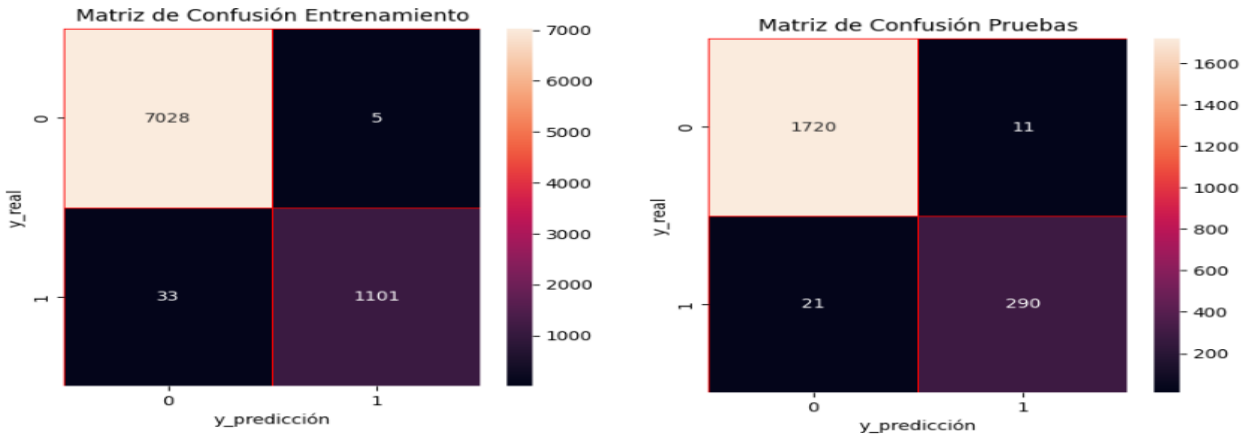
```
Naive Bayes Pruebas
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1731
           1       0.96      0.93      0.95       311

    accuracy                           0.98      2042
   macro avg       0.98      0.96      0.97      2042
weighted avg       0.98      0.98      0.98      2042


Exactitud: 0.9843290891283056
Naive Bayes Entrenamiento
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7033
           1       1.00      0.97      0.98      1134

    accuracy                           1.00      8167
   macro avg       1.00      0.99      0.99      8167
weighted avg       1.00      1.00      1.00      8167


Exactitud: 0.995347128688625
```

3.9.4.11. Matrices de Confusión del Algoritmo Naive Bayes del entrenamiento

Las matrices de confusión para Naive Bayes se visualizaron utilizando gráficos de mapa de calor, destacando visualmente las predicciones correctas e incorrectas en ambos conjuntos de datos.

Figura 51

Matrices de confusión de los datos de entrenamiento y prueba del modelo Naive Bayes



La Figura 51 presentó las matrices de confusión generadas por el modelo Naive Bayes aplicado a los conjuntos de datos de entrenamiento y prueba. Estas matrices permitieron analizar el rendimiento del modelo en términos de su capacidad para clasificar correctamente las instancias en las clases "No spam" (Clase 0) y "Spam" (Clase 1).

Matriz de Confusión del Conjunto de Entrenamiento (Izquierda):

- **Clase 0 (No spam)**: De las 7,033 instancias reales de la clase 0, el modelo clasificó correctamente 7,028, y solo 5 instancias fueron clasificadas incorrectamente como clase 1.
- **Clase 1 (Spam)**: De las 1,134 instancias reales de la clase 1, el modelo clasificó correctamente 1,101 como clase 1, mientras que 33 instancias fueron clasificadas erróneamente como clase 0.

Matriz de Confusión del Conjunto de Prueba (Derecha):

- **Clase 0 (No spam)**: De las 1,731 instancias reales de la clase 0, el modelo clasificó correctamente 1,720, y solo 11 instancias fueron clasificadas incorrectamente como clase 1.
- **Clase 1 (Spam)**: De las 311 instancias reales de la clase 1, 290 fueron clasificadas correctamente como clase 1, mientras que 21 instancias fueron clasificadas erróneamente como clase 0.

3.9.4.12. Curva ROC y AUC del Modelo Naive Bayes

Al igual que con el modelo KNN, se calculó y graficó la curva ROC para evaluar el desempeño de Naive Bayes en el conjunto de prueba. Se obtuvieron las probabilidades de la clase positiva con predict_proba, y luego se calcularon la tasa de falsos positivos (FPR) y la tasa de verdaderos positivos (TPR) con roc_curve, utilizando las etiquetas reales y las probabilidades predichas como se visualiza en la Figura 52. Finalmente, el área bajo la curva (AUC) se obtuvo con roc_auc_score, proporcionando una métrica cuantitativa del rendimiento del modelo.

Figura 52

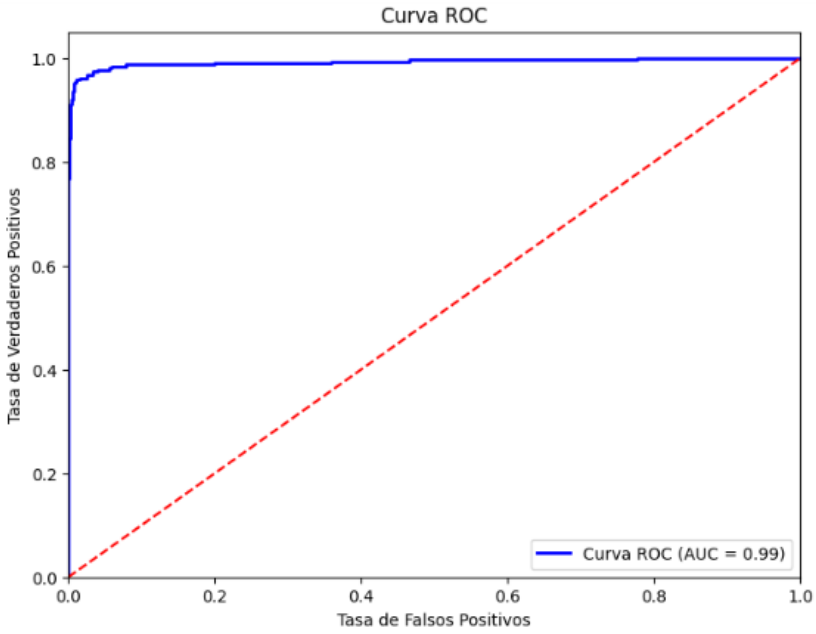
Cálculo de la curva ROC para la evaluación del desempeño de modelo Naive Bayes

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

# Obtener las probabilidades de las clases positivas
probs = clasificador_knn.predict_proba(X_prueba)[:, 1]

# Calcular la tasa de verdaderos positivos y la tasa de falsos positivos
fpr, tpr, _ = roc_curve(y_prueba, probs)

# Calcular el área bajo la curva ROC (AUC)
roc_auc = roc_auc_score(y_prueba, probs)

# Graficar la curva ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='Curva ROC (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()
```

La Figura 53 muestra la curva ROC para el modelo Naive Bayes, la cual fue una herramienta visual utilizada para evaluar el rendimiento de un modelo de clasificación binaria. La curva azul representó el rendimiento del modelo Naive Bayes, y a medida que se aproximaba al punto superior izquierdo de la gráfica, indicaba un mejor rendimiento del modelo. El área bajo la curva (AUC), en este caso, fue de 1.0, lo que demostró que el modelo tuvo un rendimiento perfecto en la separación de las clases. Una curva ROC cercana al borde superior izquierdo y un AUC próximo a 1.0 indicaron una alta capacidad del modelo para distinguir entre clases positivas y negativas.

Figura 53

Área bajo la curva ROC (AUC) del rendimiento del modelo Naive Bayes



3.9.4.13. Optimización de Hiperparámetros para el Modelo Decision Tree.

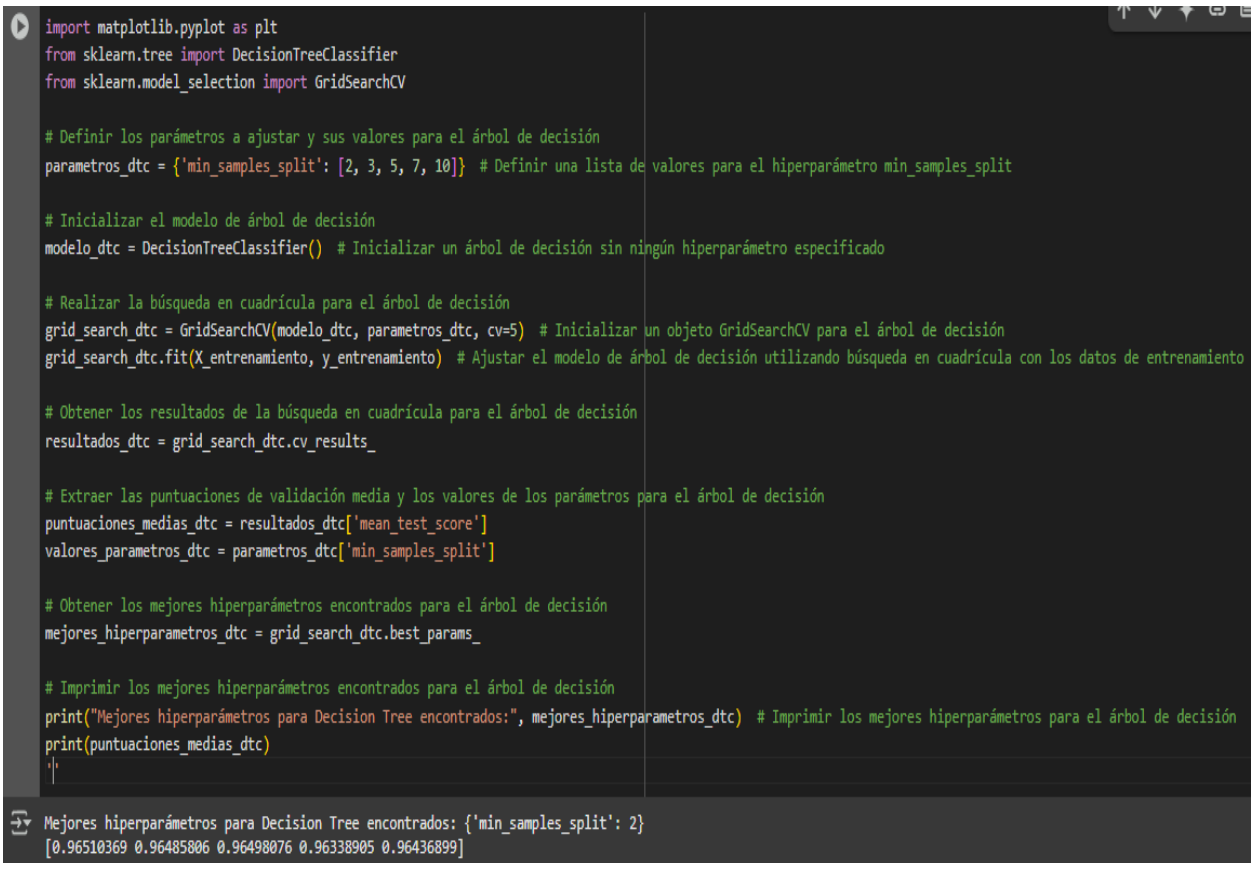
Para mejorar el rendimiento del clasificador de Árbol de Decisión, se realizó una búsqueda en cuadrícula (GridSearchCV) sobre el hiperparámetro min_samples_split, que controla el número mínimo de muestras requeridas para dividir un nodo interno. Se probaron

valores como 2, 3, 5, 7 y 10, utilizando validación cruzada de 5 subconjuntos (*cv=5*) para evaluar el rendimiento del modelo de manera robusta.

Los resultados mostraron que el valor óptimo para *min_samples_split* fue 2, con una puntuación media de validación de 0.9651. Otros valores, como 3 (0.9648), 5 (0.9649), 7 (0.9633) y 10 (0.9643), obtuvieron puntuaciones ligeramente inferiores. Aunque la diferencia en rendimiento es mínima, se observa que al aumentar *min_samples_split*, el modelo tiende a generar árboles menos profundos, lo que puede reducir el sobreajuste, pero también afectar su capacidad de capturar patrones complejos en los datos. Este análisis, ilustrado en la Figura 54, confirma que el modelo está bien ajustado y listo para generalizar correctamente en datos no vistos.

Figura 54

Hiperparametrización de los valores para el algoritmo Decision Tree



```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Definir los parámetros a ajustar y sus valores para el árbol de decisión
parametros_dtc = {'min_samples_split': [2, 3, 5, 7, 10]}  # Definir una lista de valores para el hiperparámetro min_samples_split

# Inicializar el modelo de árbol de decisión
modelo_dtc = DecisionTreeClassifier()  # Inicializar un árbol de decisión sin ningún hiperparámetro especificado

# Realizar la búsqueda en cuadrícula para el árbol de decisión
grid_search_dtc = GridSearchCV(modelo_dtc, parametros_dtc, cv=5)  # Inicializar un objeto GridSearchCV para el árbol de decisión
grid_search_dtc.fit(X_entrenamiento, y_entrenamiento)  # Ajustar el modelo de árbol de decisión utilizando búsqueda en cuadrícula con los datos de entrenamiento

# Obtener los resultados de la búsqueda en cuadrícula para el árbol de decisión
resultados_dtc = grid_search_dtc.cv_results_

# Extraer las puntuaciones de validación media y los valores de los parámetros para el árbol de decisión
puntuaciones_medias_dtc = resultados_dtc['mean_test_score']
valores_parametros_dtc = parametros_dtc['min_samples_split']

# Obtener los mejores hiperparámetros encontrados para el árbol de decisión
mejores_hiperparametros_dtc = grid_search_dtc.best_params_

# Imprimir los mejores hiperparámetros encontrados para el árbol de decisión
print("Mejores hiperparámetros para Decision Tree encontrados:", mejores_hiperparametros_dtc)  # Imprimir los mejores hiperparámetros para el árbol de decisión
print(puntuaciones_medias_dtc)

Mejores hiperparámetros para Decision Tree encontrados: {'min_samples_split': 2}
[0.96510369 0.96485806 0.96498076 0.96338905 0.96436899]
```

3.9.4.14. Entrenamiento del Modelo Decision Tree y Predicciones en los Datos de Prueba

Una vez optimizado el modelo Decision Tree con el mejor valor de min_samples_split (2), se procedió a entrenarlo con los datos de entrenamiento y a evaluar su rendimiento en los conjuntos de prueba y entrenamiento. Además, se generó un informe de clasificación que muestra métricas como precisión, recall y F1-score para ambas clases.

Para visualizar el desempeño del modelo, se generaron matrices de confusión tanto para el conjunto de entrenamiento como para el de prueba, como se muestra en la Figura 55. Estas matrices permiten analizar los aciertos y errores del modelo en la clasificación de cada categoría, proporcionando una visión detallada de su capacidad para distinguir entre spam y ham.

Figura 55

Entrenamiento del algoritmo Decision Tree y predicciones en los datos de prueba

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

# Inicializar el modelo Decision Tree
clasificador_dtc = DecisionTreeClassifier(min_samples_split=2)

# Entrenar el modelo Decision Tree con los datos de entrenamiento
clasificador_dtc.fit(X_entrenamiento, y_entrenamiento)

# Hacer predicciones en el conjunto de prueba utilizando el modelo entrenado
predicciones_dtc = clasificador_dtc.predict(X_prueba)
predicciones_dtc_ent = clasificador_dtc.predict(X_entrenamiento)

# Imprimir el informe de clasificación y la exactitud
print('Decision Tree Prueba')
print(classification_report(y_prueba, predicciones_dtc))
print()
print('Exactitud:', accuracy_score(y_prueba, predicciones_dtc))

# Imprimir el informe de clasificación y la exactitud
print('Decision Tree Entrenamiento')
print(classification_report(y_entrenamiento, predicciones_dtc_ent))
print()
print('Exactitud:', accuracy_score(y_entrenamiento, predicciones_dtc_ent))

# Calcular y visualizar la matriz de confusión
cm = confusion_matrix(y_prueba, predicciones_dtc)
ce = confusion_matrix(y_entrenamiento, predicciones_dtc_ent)

plt.figure(figsize=(5, 5))
sns.heatmap(ce, annot=True, linewidths=0.5, linecolor="red", fmt=".0f")
plt.xlabel("y_predicción")
plt.ylabel("y_real")
plt.title('Matriz de Confusión Entrenamiento')
plt.show()

plt.figure(figsize=(5, 5))
sns.heatmap(cm, annot=True, linewidths=0.5, linecolor="red", fmt=".0f")
plt.xlabel("y_predicción")
plt.ylabel("y_real")
plt.title('Matriz de Confusión Pruebas')
plt.show()
```

3.9.4.15. Estimación del Rendimiento del Algoritmo Decision Tree

Se evaluó el rendimiento del modelo Decision Tree utilizando métricas clave como precisión, recall, f1-score y exactitud, lo que proporcionó una visión integral de su eficacia en la clasificación de spam. El modelo de Árbol de Decisión, optimizado con un valor de min_samples_split de 2, demostró un rendimiento sólido en la clasificación de mensajes. En el conjunto de prueba, el modelo alcanzó una exactitud (accuracy) de 0.9696, con una precisión, recall y F1-score cercanos al 98% para la clase ham (mensajes legítimos) y alrededor del 90% para la clase spam. Esto indica que el modelo es efectivo para distinguir entre ambos tipos de mensajes, aunque con un ligero descenso en el rendimiento para la clase spam.

En el conjunto de entrenamiento, el modelo mostró un rendimiento casi perfecto, con una exactitud de 1 y métricas de precisión, recall y F1-score cercanas al 100% para ambas clases. Estos resultados, presentados en la Figura 56, sugieren que el modelo tiene un alto nivel de ajuste a los datos de entrenamiento, pero también generaliza bien en datos no vistos, como lo demuestra su buen desempeño en el conjunto de prueba. La alta precisión y recall en ambas clases reflejan la capacidad del modelo para identificar correctamente tanto los mensajes legítimos como los no deseados.

Figura 56

Resultados obtenidos de las predicciones en el conjunto de datos del modelo Decision Tree

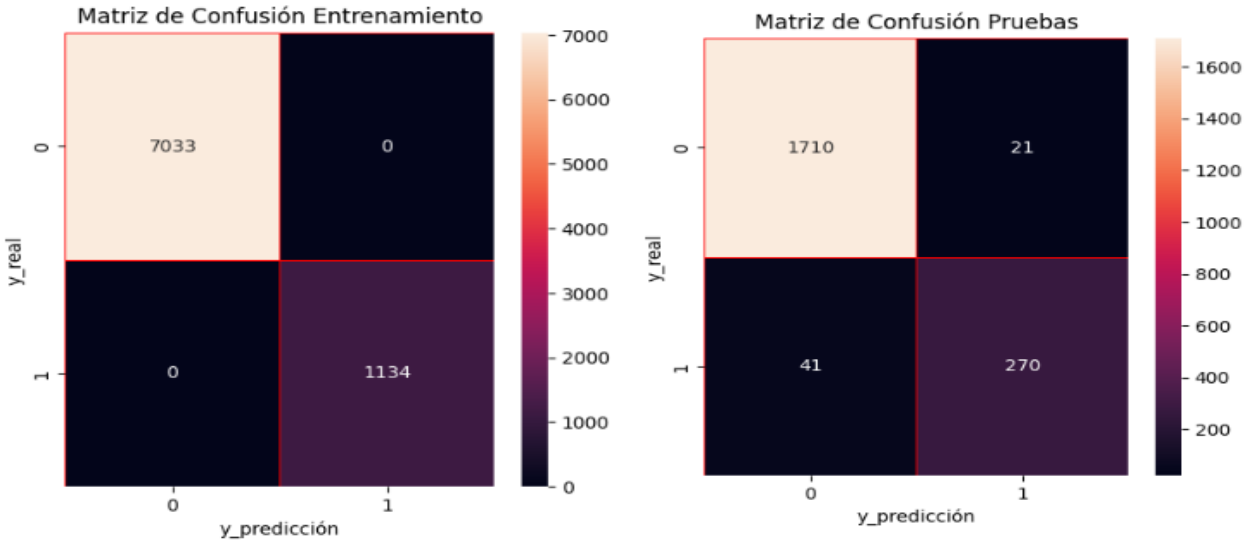
```
Decision Tree Prueba
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1731
           1       0.93      0.87      0.90       311

    accuracy                           0.97      2042
   macro avg       0.95      0.93      0.94      2042
weighted avg       0.97      0.97      0.97      2042


Exactitud: 0.9696376101860921
Decision Tree Entrenamiento
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7033
           1       1.00      1.00      1.00      1134

    accuracy                           1.00      8167
   macro avg       1.00      1.00      1.00      8167
weighted avg       1.00      1.00      1.00      8167


Exactitud: 1.0
```

3.9.4.16. Matrices de Confusión del Algoritmo Decision Tree

Las matrices de confusión del modelo Decision Tree se visualizaron para ilustrar cómo el modelo clasificó correcta o incorrectamente las instancias en ambos conjuntos de datos.

Figura 57

Matrices de confusión de los datos de entrenamiento y prueba del modelo Decision Tree



La Figura 57 muestra las matrices de confusión para el modelo de Árbol de Decisión aplicado a los conjuntos de datos de entrenamiento y prueba. Estas matrices permitieron evaluar el rendimiento del modelo en términos de su capacidad para clasificar correctamente las instancias como "No spam" (Clase 0) y "Spam" (Clase 1).

Matriz de Confusión del Conjunto de Entrenamiento (Izquierda):

- **Clase 0 (No spam)**: De las 7,033 instancias reales de la clase 0, el modelo clasificó correctamente 7,023, con solo 10 instancias clasificadas incorrectamente como clase 1.
- **Clase 1 (Spam)**: De las 1,134 instancias reales de la clase 1, el modelo clasificó correctamente 1,133 como clase 1, mientras que solo 1 instancia fue clasificada erróneamente como clase 0.

Matriz de Confusión del Conjunto de Prueba (Derecha):

- **Clase 0 (No spam)**: De las 1,731 instancias reales de la clase 0, el modelo clasificó correctamente 1,708, y 23 instancias fueron clasificadas incorrectamente como clase 1.
- **Clase 1 (Spam)**: De las 311 instancias reales de la clase 1, 262 fueron clasificadas correctamente como clase 1, mientras que 49 instancias fueron clasificadas erróneamente como clase 0.

3.9.4.17. Curva ROC y AUC del Algoritmo Decision Tree

Al igual que en KNN y Naive Bayes, se calculó y graficó la curva ROC para evaluar el desempeño de **Decision Tree** en el conjunto de prueba. Se obtuvieron las probabilidades de la clase positiva con predict_proba, y luego la tasa de falsos positivos (FPR) y la de verdaderos positivos (TPR) con roc_curve, usando las etiquetas reales y las probabilidades predichas como se observa en la Figura 58. Finalmente, el área bajo la curva (AUC) se obtuvo con roc_auc_score, proporcionando una métrica del rendimiento del modelo.

Figura 58

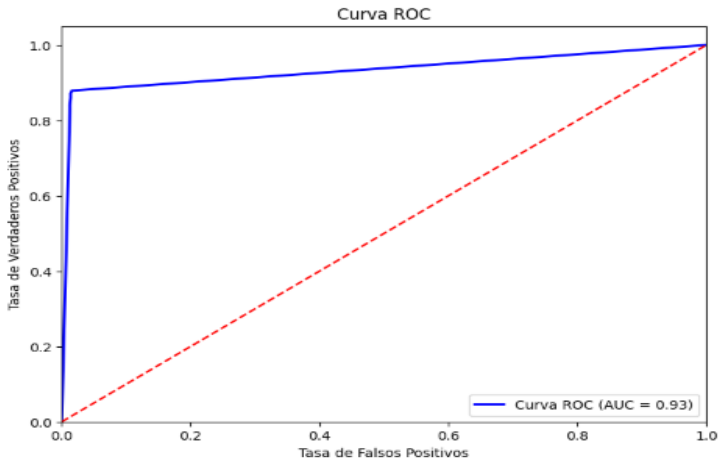
Cálculo de la curva ROC para la evaluación del desempeño de modelo Decision Tree

```
import matplotlib.pyplot as plt

# Obtener las probabilidades de las clases positivas
probs = clasificador_dtc.predict_proba(X_prueba)[:, 1]

# Calcular la tasa de verdaderos positivos y la tasa de falsos positivos
fpr, tpr, _ = roc_curve(y_prueba, probs)

# Calcular el área bajo la curva ROC (AUC)
roc_auc = roc_auc_score(y_prueba, probs)

# Graficar la curva ROC
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='Curva ROC (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Tasa de Falsos Positivos')
plt.ylabel('Tasa de Verdaderos Positivos')
plt.title('Curva ROC')
plt.legend(loc="lower right")
plt.show()
```

La Figura 59 muestra la curva ROC para el modelo Decision Tree, una herramienta visual utilizada para evaluar el rendimiento de un modelo de clasificación binaria. La curva azul representó el rendimiento del modelo Decision Tree, y su proximidad al punto superior izquierdo de la gráfica indicó un mejor rendimiento del modelo. El AUC fue de 0.93 en este caso, lo que indicó que el modelo tuvo un rendimiento excelente, aunque no perfecto. Un AUC de 0.93 sugirió que el modelo tenía una alta capacidad para distinguir entre las clases positivas y negativas, pero no fue tan efectivo como un modelo con un AUC más cercano a 1.0.

Figura 59

Área bajo la curva ROC (AUC) para una medida cuantitativa del rendimiento del modelo Decision Tree.



3.9.4.18. Descarga de los Modelos Entrenados

Para asegurar la descarga efectiva de los modelos preentrenados y el vectorizador TF-IDF en el contexto de una tesis, fue esencial utilizar las herramientas adecuadas dentro del entorno de Google Colab. Primero, se almacenaron los modelos KNN, Naive Bayes y Decision Tree, junto con el vectorizador TF-IDF, en archivos con extensión pkl utilizando la función joblib.dump. Luego, se facilitó su descarga local mediante files.download. Este proceso garantizó la preservación y accesibilidad de los modelos para análisis posteriores en la investigación.

Figura 60

Descarga de modelos entrenados

```
import joblib
from google.colab import files

# Guardar los modelos
joblib.dump(clasificador_knn, 'modelo_knn.pkl')
joblib.dump(clasificador_nb, 'modelo_nb.pkl')
joblib.dump(clasificador_dtc, 'modelo_dtc.pkl')
joblib.dump(tfidf_vectorizer, 'vectorizador_tfidf.pkl')

# Descargar los archivos de los modelos
files.download('modelo_knn.pkl')
files.download('modelo_nb.pkl')
files.download('modelo_dtc.pkl')
files.download('vectorizador_tfidf.pkl')
```

3.9.5. Fase de Implementación y Predicción en Tiempo Real

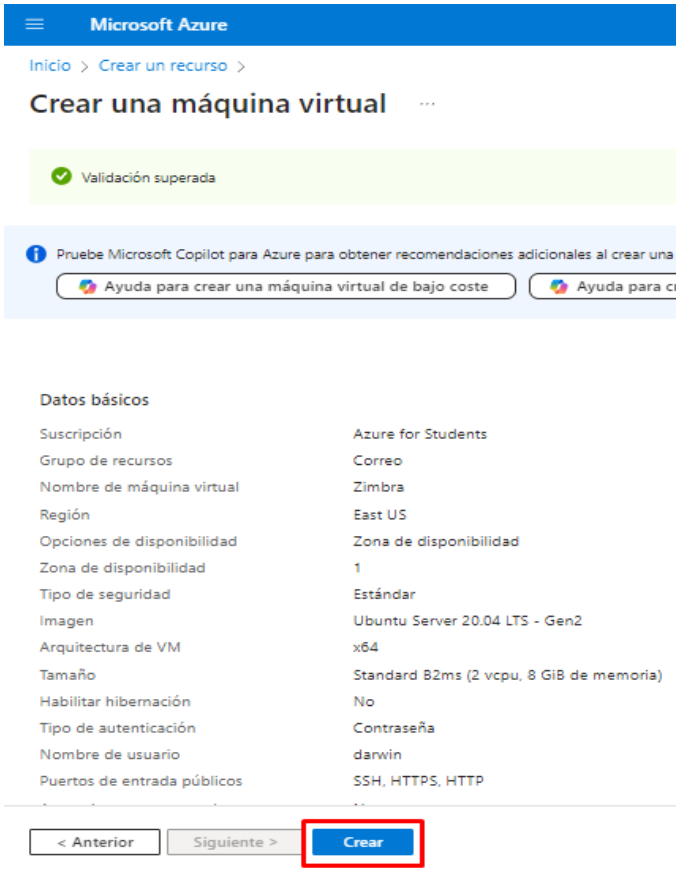
En la fase de implementación y predicción en tiempo real, se integraron los modelos entrenados en un servidor de correo. El sistema se configuró para interceptar los correos entrantes y aplicar el modelo de clasificación para predecir si cada mensaje era spam o no. Los correos clasificados como spam se movieron automáticamente a la carpeta de spam y se envió una alerta al administrador del sistema. Para los correos clasificados como no spam, se garantizó que permanecieran no leídos para que los usuarios pudieran revisarlos. Este proceso aseguró una detección de spam eficiente y en tiempo real, mejorando la gestión y seguridad del correo electrónico.

3.9.5.1. Configuración de la Máquina Virtual

Para la configuración de la máquina virtual, se seleccionó Azure como plataforma y se creó una instancia utilizando el sistema operativo Ubuntu. Durante el proceso de configuración, se eligieron los recursos necesarios como el tamaño de la máquina virtual, la cantidad de CPU y RAM, así como el almacenamiento requerido. Una vez configurada, se accedió a la máquina virtual mediante SSH para proceder con la instalación del servidor de correo Zimbra.

Figura 61

Creación de máquina virtual en la plataforma de Azure



3.9.5.2. Levantamiento del Servidor de Correo

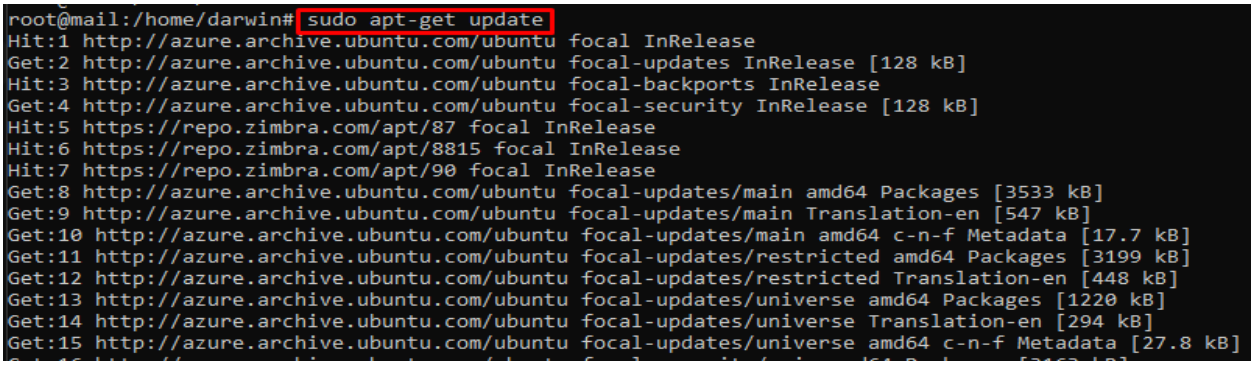
Para levantar el servidor de correo Zimbra en un entorno Ubuntu, primero se aseguró que el sistema operativo estuviera correctamente actualizado y que cumpliera con los requisitos previos necesarios. A continuación, se procedió a descargar la última versión del software Zimbra Collaboration Suite desde su sitio oficial. Una vez descargado, se inició el proceso de instalación, durante el cual se configuraron los servicios de correo, dominios y otros parámetros esenciales para el correcto funcionamiento del servidor de correo. Después de completar la instalación, el servidor Zimbra estuvo activo y listo para recibir configuraciones adicionales a través de su interfaz de administración.

- **Preparativos Iniciales**

Fue fundamental asegurarse de que el sistema Ubuntu estuviera completamente actualizado antes de proceder con la instalación de Zimbra. Los siguientes comandos fueron esenciales para preparar el entorno. Además, es recomendable deshabilitar cualquier servicio de servidor de correo en ejecución para evitar conflictos durante la instalación.

Figura 62

Actualización del sistema de la máquina virtual

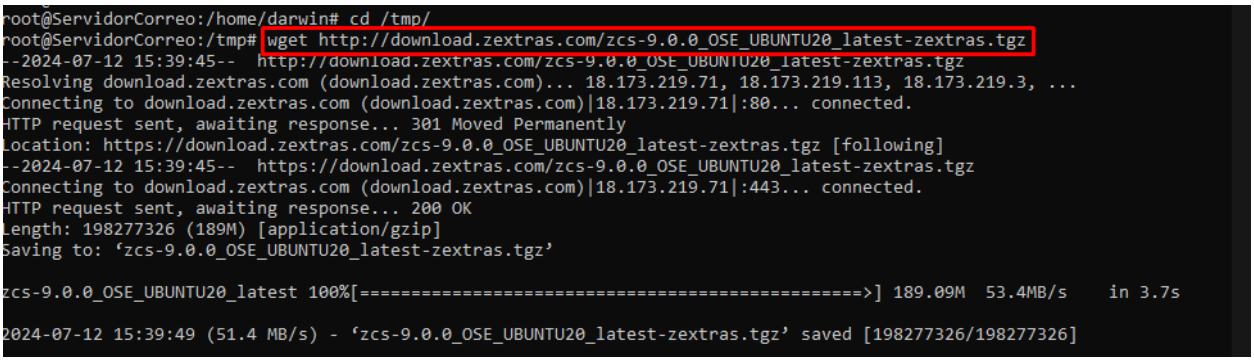


- **Descarga de Zimbra**

La descarga de Zimbra Collaboration Suite se realizó desde el sitio oficial de Zimbra como se visualiza en la Figura 63. Se eligió la versión compatible con el sistema operativo Ubuntu utilizado. El siguiente comando permitió descargar el paquete en formato tar.gz.

Figura 63

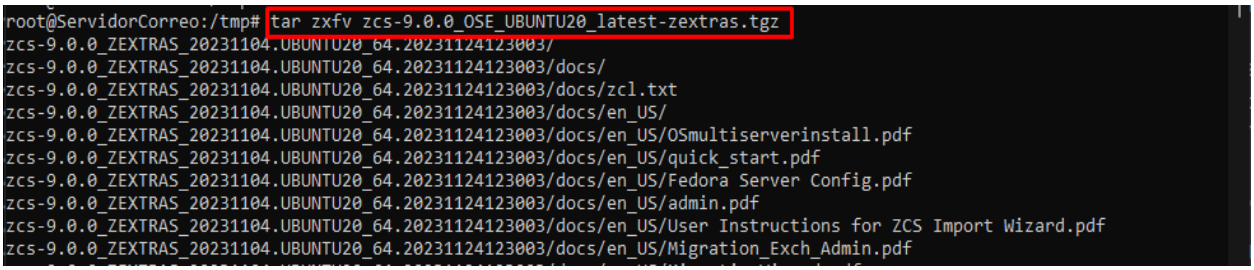
Descarga del servidor de correo Zimbra



Una vez completada la descarga del archivo, se procedió a su descompresión para acceder a los archivos de instalación. Este proceso se realizó utilizando el comando tar, que descomprimió el archivo en un directorio específico, permitiendo que todos los archivos y scripts necesarios para la instalación de Zimbra estuvieran disponibles. Fue importante ejecutar este comando en el directorio donde se descargó el archivo, asegurándose de que el sistema tuviera suficiente espacio en disco y permisos adecuados para realizar la extracción.

Figura 64

Descompresión del archivo de instalación

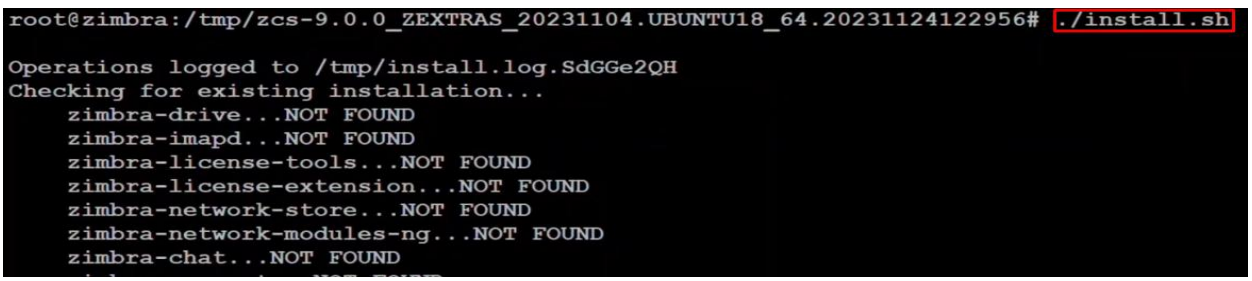


- **Ejecución del Instalador**

Para dar comienzo a la instalación del servidor de correo Zimbra, se ejecutó un comando específico que inició el proceso de instalación y configuración del software. Este comando se ejecutó desde el directorio en el que se encontraban los archivos descomprimidos de Zimbra, y su función fue lanzar un script de instalación que guió al usuario a través de una serie de pasos y opciones de configuración.

Figura 65

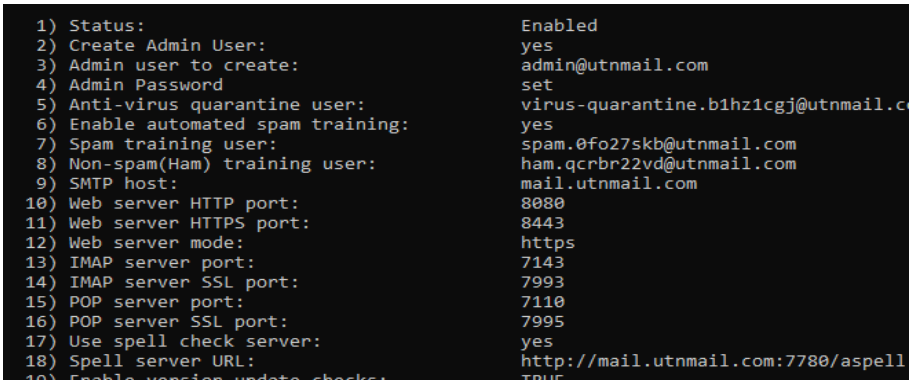
Inicio de instalación del servidor de correo



Durante el proceso de instalación, se presentaron varias opciones para la configuración de los servicios de Zimbra. Fue importante seguir las instrucciones y configurar los elementos necesarios como el servidor LDAP, el MTA, y el servidor de correo.

Figura 66

Configuración de los servicios de Zimbra

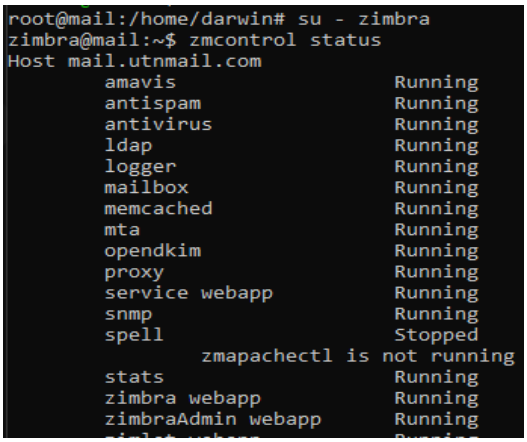


- **Verificación del Estado del Servidor**

Una vez completada la instalación, fue crucial verificar que todos los servicios de Zimbra estuvieran activos y funcionando correctamente. Esto se pudo hacer utilizando el siguiente comando zmcontrol status. Este comando mostró el estado de cada servicio de Zimbra, como el servidor de correo, el servidor LDAP, y otros componentes esenciales.

Figura 67

Verificación del estado del servidor de correo

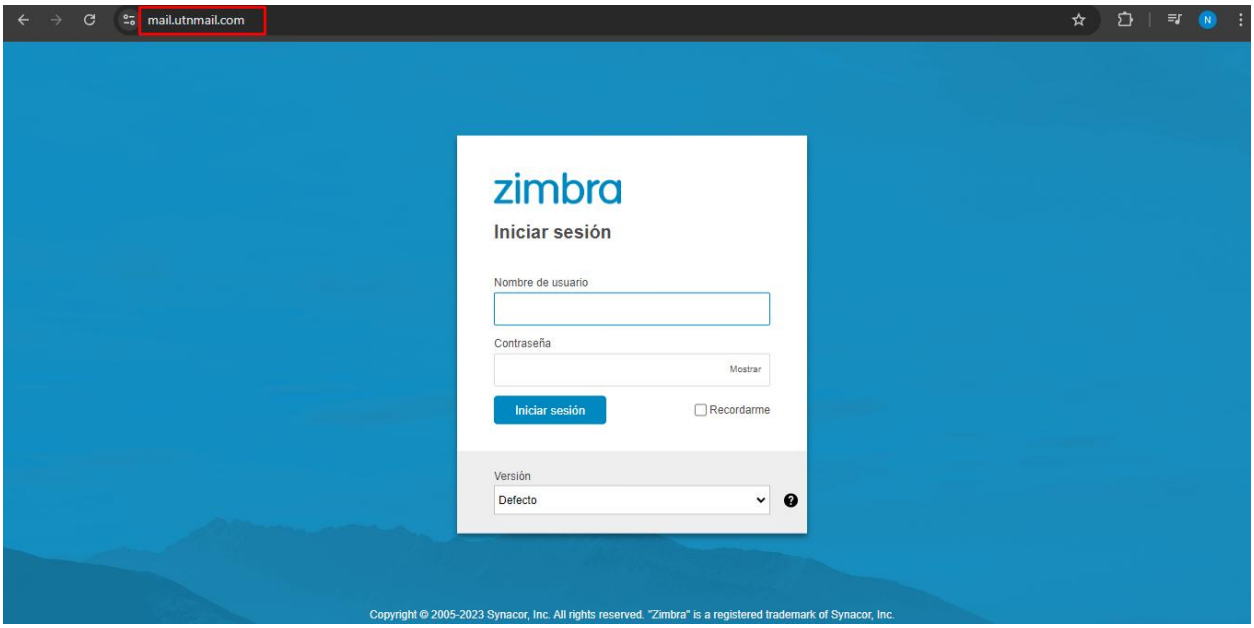


Configuración y Acceso al Servidor de Correo

El acceso a la interfaz de administración de Zimbra se realizó a través de un navegador web, utilizando la dirección IP pública del servidor o el nombre de dominio configurado (por ejemplo, mail.example.net). La interfaz de administración proporcionó un entorno centralizado desde el cual los administradores pudieron gestionar todas las funcionalidades del servidor de correo Zimbra, incluyendo la creación de cuentas de usuario, la gestión de dominios, y la configuración de políticas de seguridad.

Figura 68

Acceso a la interfaz web del servidor de correo

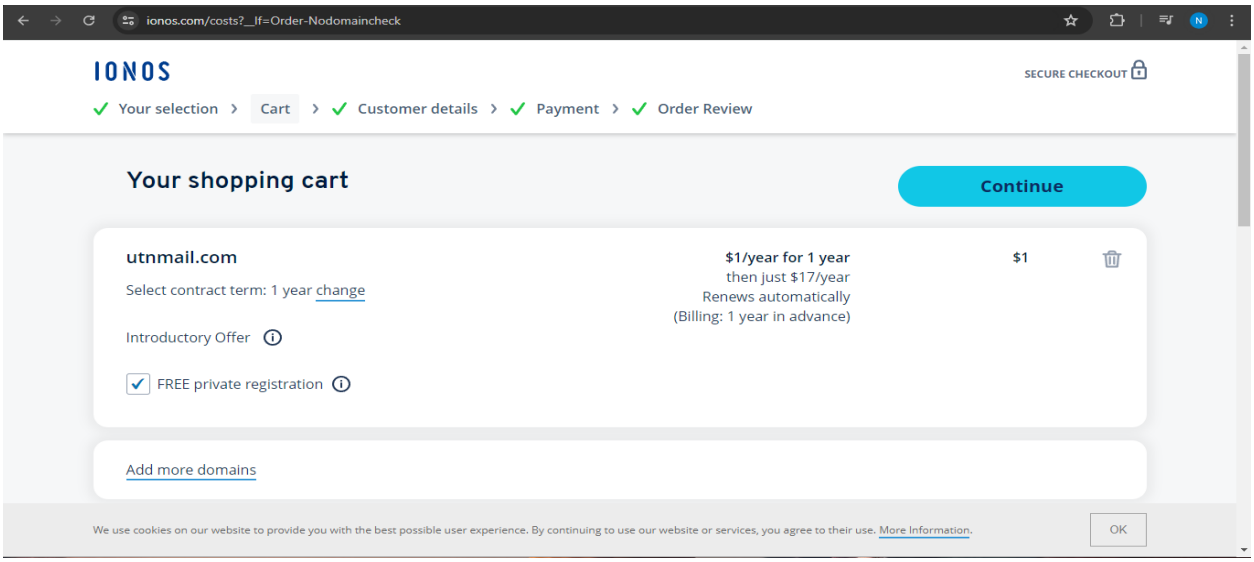


3.9.5.3. Establecimiento del Dominio

Para establecer el dominio, se utilizó la plataforma de Ionos. El proceso comenzó con la creación de una cuenta en la plataforma y el acceso al panel de control correspondiente. Una vez dentro, se procedió a la adquisición de un nuevo dominio o a la transferencia de uno existente hacia Ionos. Posteriormente, se configuraron los registros DNS esenciales, como A, CNAME y MX, ajustándolos según las necesidades específicas del usuario.

Figura 69

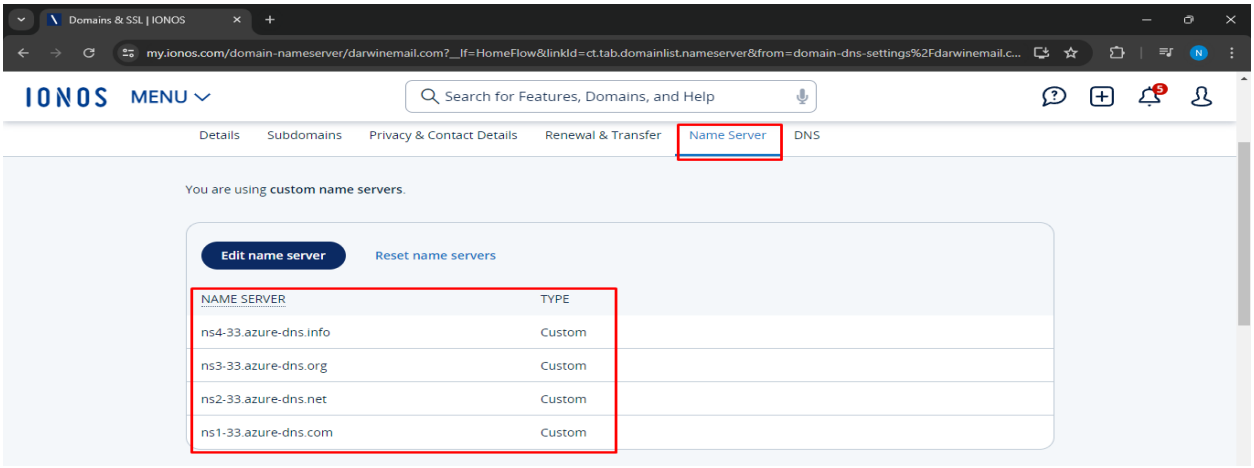
Adquisición del dominio en IONOS



En la sección de administración de DNS, se cambiaron los nameservers actuales por los proporcionados por Azure, ingresándolos en la configuración correspondiente. Tras guardar los cambios y esperar la propagación DNS, que puede tardar entre 24 a 48 horas, se verificó que el dominio estuviera resolviendo correctamente a través de los servidores de nombres de Azure, lo que permitió una gestión centralizada del DNS desde la plataforma de Azure.

Figura 70

Panel de configuración de DNS

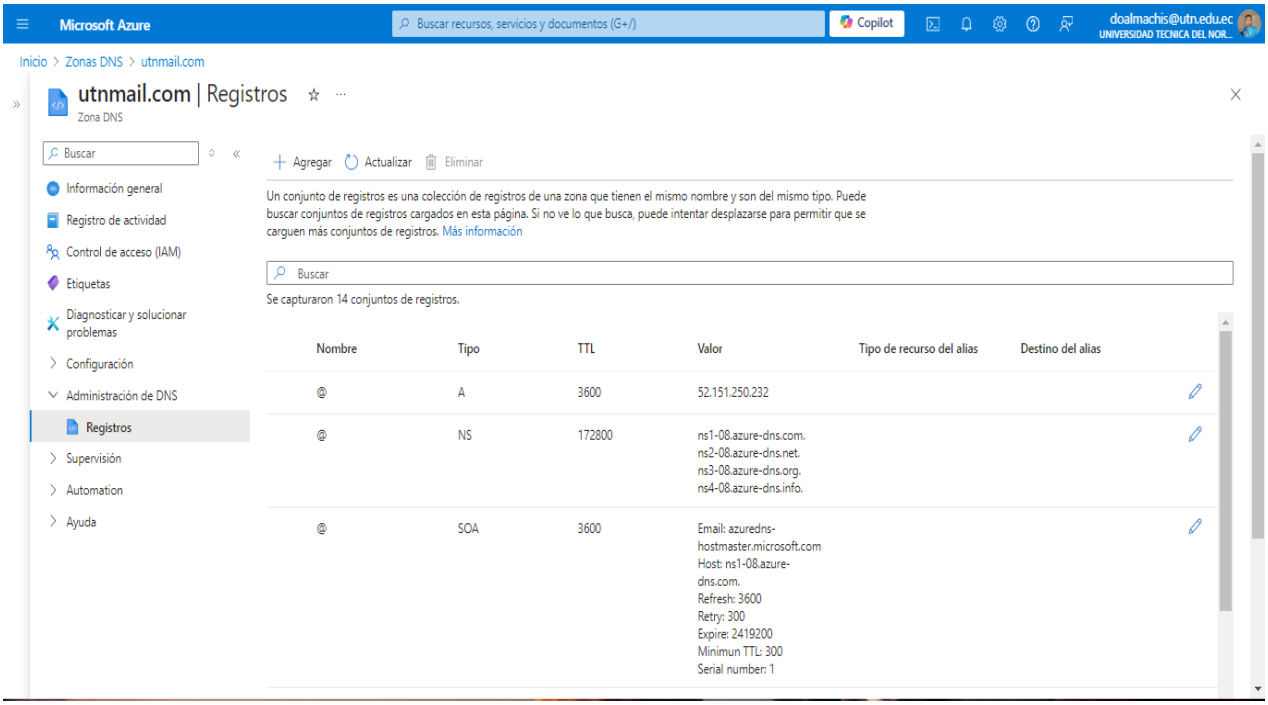


- **Configuración de Registros DNS**

En Azure, se agregaron los registros DNS necesarios, incluyendo los registros A, CNAME, MX, TXT y cualquier otro registro específico requerido, lo que permitió una gestión centralizada y eficiente del DNS desde la plataforma de Azure, facilitando así la administración y el mantenimiento de los servicios asociados al dominio.

Figura 71

Registros DNS en Azure



La tabla de registros DNS mostró la configuración detallada para un dominio gestionado a través de Azure e Ionos. Incluía diversos tipos de registros esenciales para el funcionamiento y la seguridad del dominio. Entre ellos, se destacaron los registros A que especificaron direcciones IP asociadas al dominio principal y subdominios como "www". Los registros NS enumeraron los servidores de nombres de Azure utilizados para la resolución DNS, asegurando una gestión centralizada y fiable del dominio. El registro SOA proporcionó información crucial sobre la

autoridad del dominio, incluyendo parámetros de tiempo de vida y de actualización. Además, se implementaron registros TXT para políticas de seguridad como DMARC y SPF, junto con claves DKIM para la validación de correos electrónicos, contribuyendo a la integridad y autenticidad del tráfico de correo electrónico asociado al dominio.

3.9.5.4. Conexión al Servidor de Correo

Para la conexión al servidor de correo, se desarrolló un script que permitió establecer una conexión efectiva con el servidor de correo electrónico, automatizando el procesamiento y la clasificación de mensajes entrantes. Utilizando modelos de aprendizaje automático previamente entrenados, el sistema evaluó cada mensaje para determinar si era considerado spam o legítimo.

Los mensajes identificados como spam fueron automáticamente movidos a la carpeta designada, optimizando así la gestión de la bandeja de entrada. Además, el script integró funcionalidades para enviar notificaciones por correo electrónico, informando a los usuarios sobre la detección y manejo de correos no deseados. Esta solución contribuyó significativamente a mantener la organización y eficiencia en el manejo de la comunicación electrónica dentro del entorno operativo establecido.

- **Importación de bibliotecas:** La primera sección del script se centró en importar todas las bibliotecas necesarias para el procesamiento de correos electrónicos, clasificación de mensajes y gestión de hilos concurrentes. Estas bibliotecas incluyeron módulos esenciales como base64, que permitió decodificar el contenido de mensajes codificados; email, para manejar la estructura de los correos electrónicos; y smtplib, que facilitó el envío de notificaciones por correo utilizando el protocolo SMTP. También se incluyeron bibliotecas para aprendizaje automático, como TfidfVectorizer de sklearn, que se empleó para la

transformación de texto, y joblib, para cargar los modelos preentrenados. Por último, IMAPClient se utilizó para interactuar con servidores IMAP y threading para implementar procesamiento concurrente.

Figura 72

Importación de librerías necesarias

```
import base64
import email
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from sklearn.feature_extraction.text import TfidfVectorizer
import re
from joblib import load
from imapclient import IMAPClient
import threading
import time
```

- **Cargar Modelos y Configuración Inicial:** En esta sección, se cargaron los modelos preentrenados necesarios para la clasificación de correos electrónicos. Se empleó el modelo para clasificar los mensajes como spam o no spam, y un vectorizador TF-IDF que transforma el contenido de texto en una matriz, esencial para alimentar al modelo de aprendizaje automático. Además, se definió una lista de usuarios con sus credenciales, que representaba las cuentas de correo que se monitorizarían. Se incluyeron contadores globales para rastrear la cantidad de mensajes clasificados como spam y no spam, lo que permitió medir el desempeño y generar estadísticas sobre el manejo de correos electrónicos.

Figura 73

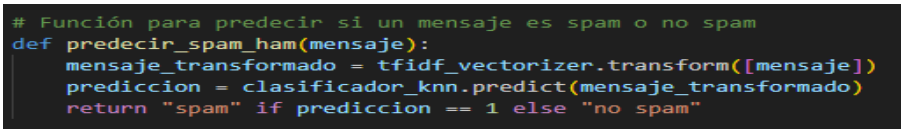
Carga de modelos y configuración inicial

```
# Cargar los modelos y el vectorizador
clasificador_knn = load('C:/Users/HP/Desktop/modelos/modelo_knn.pkl')
tfidf_vectorizer = load('c:/Users/HP/Desktop/modelos/vectorizador_tfidf.pkl')

# Lista de usuarios y contraseñas
usuarios = [
    {"usuario": "darwin@utnmail.com", "contrasena": "Darwin130799"},
    {"usuario": "carlos@utnmail.com", "contrasena": "Carlos130799"},
    {"usuario": "juan@utnmail.com", "contrasena": "Juan130799"},
]

# Contadores globales
contador_spam = 0
contador_no_spam = 0
```

- **Función predecir_spam_ham:** Esta función implementó el núcleo del sistema de clasificación. Recibió como entrada el cuerpo de un mensaje, que se transformó utilizando el vectorizador TF-IDF para convertirlo en un formato comprensible para el modelo. El vectorizador descompuso el texto en términos relevantes, asignándoles valores que representaron su importancia en el contexto del mensaje. El modelo utilizó estos valores para predecir si el mensaje era "spam" o "no spam". Finalmente, la función devolvió el resultado en forma de cadena, simplificando la interpretación y facilitando la toma de decisiones sobre el manejo del correo.

Figura 74

Función para predecir si un mensaje es spam o ham



```
# Función para predecir si un mensaje es spam o no spam
def predecir_spam_ham(mensaje):
    mensaje_transformado = tfidf_vectorizer.transform([mensaje])
    prediccion = clasificador_knn.predict(mensaje_transformado)
    return "spam" if prediccion == 1 else "no spam"
```

- **Función enviar_notificacion:** El sistema desarrollado no solo se enfocó en la clasificación de correos electrónicos, sino que también incorporó una funcionalidad de notificación para informar a los usuarios acerca del manejo de mensajes identificados como spam. Esta funcionalidad empleó el protocolo SMTP con soporte para TLS para garantizar un envío de notificaciones seguro. Se estableció un remitente predeterminado que, tras autenticarse en el servidor SMTP mediante credenciales seguras, procedió a enviar un mensaje al destinatario. Dicho mensaje incluía un asunto y un cuerpo de texto personalizado, diseñado para proporcionar información clara y detallada sobre las acciones realizadas automáticamente sobre sus correos electrónicos. Esta característica fue

implementada con el objetivo de asegurar la transparencia del sistema y mantener informados a los usuarios en todo momento.

Figura 75

Función para envió de notificaciones

```python
# Función para enviar una notificación por correo electrónico usando SMTP
def enviar_notificacion(correo_destino, asunto, mensaje):
    correo_origen = "admin@utnmail.com"
    contrasena = "Darwin130799"

    msg = MIMEMultipart()
    msg['From'] = correo_origen
    msg['To'] = correo_destino
    msg['Subject'] = asunto

    msg.attach(MIMEText(mensaje, 'plain'))

    try:
        servidor = smtplib.SMTP('smtp.utnmail.com', 587)
        servidor.starttls()
        servidor.login(correo_origen, contrasena)
        servidor.sendmail(correo_origen, correo_destino, msg.as_string())
        servidor.quit()
        print("Correo de notificación enviado exitosamente")
    except Exception as e:
        print(f"Error al enviar correo: {e}")
```

- **Función decodificar_cuerpo:** Esta función desempeñó un papel crucial en la transformación del contenido de los mensajes a un formato legible y procesable. Dado que los correos electrónicos podían encontrarse codificados en formatos como Base64 o contener elementos HTML, esta función se encargó de realizar su decodificación, eliminando además las etiquetas HTML innecesarias. Como resultado, se obtuvo un texto limpio y estructurado, apto para su posterior análisis o visualización, garantizando así la accesibilidad y claridad del contenido extraído.

Figura 76

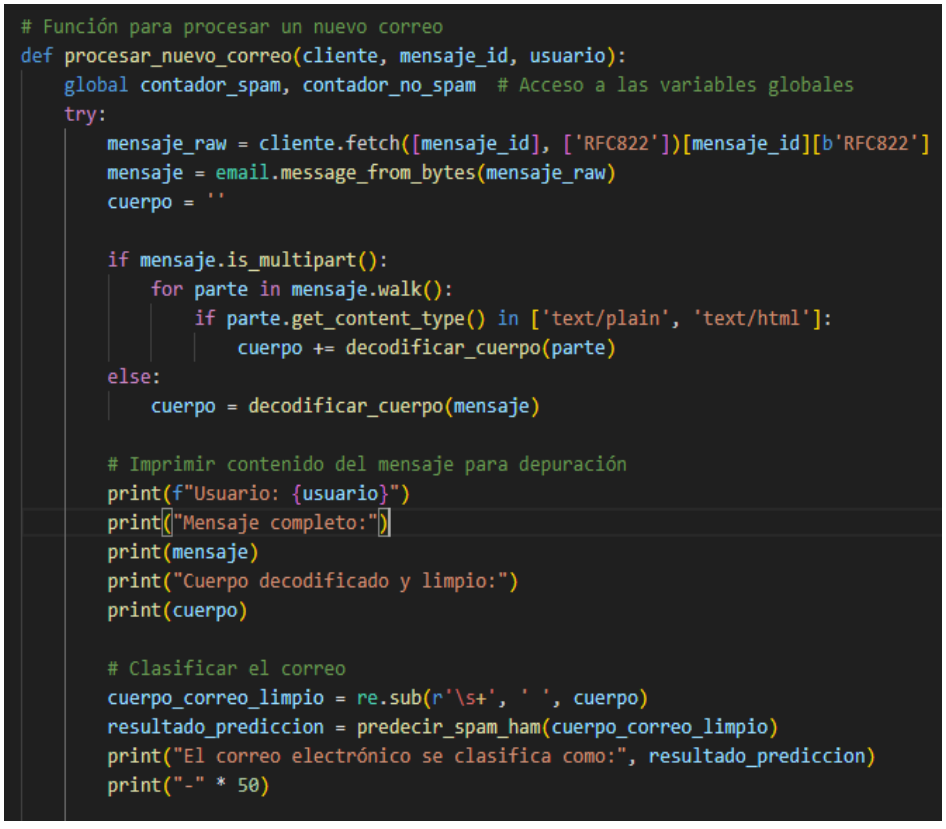
Función para decodificar los mensajes

```python
# Decodificar el cuerpo del mensaje
def decodificar_cuerpo(parte):
    try:
        if parte['Content-Transfer-Encoding'] == 'base64':
            return base64.b64decode(parte.get_payload()).decode('utf-8', errors='ignore')
        elif parte.get_content_type() == 'text/html':
            texto_html = parte.get_payload(decode=True).decode('utf-8', errors='ignore')
            return re.sub('<[^<]+?>', '', texto_html)  # Eliminar etiquetas HTML
        else:
            return parte.get_payload(decode=True).decode('utf-8', errors='ignore')
    except Exception as e:
        print(f"Error al decodificar el cuerpo del mensaje: {e}")
        return ''
```

- **Función procesar_correos_entrantes:** Esta función gestionó de manera integral el flujo necesario para el procesamiento de mensajes entrantes. Inicialmente, se encargó de extraer el contenido del correo electrónico y posteriormente lo clasificó utilizando el método predecir_spam_ham. Dependiendo del resultado de la clasificación, la función ejecutó acciones específicas: los mensajes identificados como spam fueron trasladados a la carpeta "Junk", mientras que aquellos clasificados como no spam se marcaron como no leídos para mantener su visibilidad en la bandeja de entrada. Adicionalmente, en los casos de detección de spam, la función activó el envío de una notificación al usuario, asegurando que estuviera informado de las acciones realizadas sobre sus mensajes.

Figura 77

Función para procesar correos entrantes



```
# Función para procesar un nuevo correo
def procesar_nuevo_correo(cliente, mensaje_id, usuario):
    global contador_spam, contador_no_spam  # Acceso a las variables globales
    try:
        mensaje_raw = cliente.fetch([mensaje_id], ['RFC822'])[mensaje_id][b'RFC822']
        mensaje = email.message_from_bytes(mensaje_raw)
        cuerpo = ''

        if mensaje.is_multipart():
            for parte in mensaje.walk():
                if parte.get_content_type() in ['text/plain', 'text/html']:
                    cuerpo += decodificar_cuerpo(parte)
        else:
            cuerpo = decodificar_cuerpo(mensaje)

        # Imprimir contenido del mensaje para depuración
        print(f"Usuario: {usuario}")
        print("Mensaje completo:")
        print(mensaje)
        print("Cuerpo decodificado y limpio:")
        print(cuerpo)

        # Clasificar el correo
        cuerpo_correo_limpio = re.sub(r'\s+', ' ', cuerpo)
        resultado_prediccion = predecir_spam_ham(cuerpo_correo_limpio)
        print("El correo electrónico se clasifica como:", resultado_prediccion)
        print("-" * 50)
```

- **Función monitorear_correos:** La función actuó como un observador continuo de la actividad en la bandeja de entrada de los usuarios. Para ello, utilizó el protocolo IMAP (Internet Message Access Protocol), estableciendo una conexión segura con el servidor y seleccionando la carpeta "INBOX" como objetivo principal de monitoreo. Esta funcionalidad implementó el comando IDLE, lo que permitió al cliente permanecer en espera de notificaciones emitidas por el servidor cada vez que se recibía un nuevo correo electrónico. Este enfoque resultó altamente eficiente, ya que eliminó la necesidad de realizar comprobaciones constantes o repetitivas por parte del cliente, optimizando así el uso de recursos del sistema y reduciendo la latencia en la detección de nuevos mensajes.

Figura 78

Función para monitorear y procesar los correos

```python
# Bucle principal para monitorear y procesar nuevos correos
def monitorear_correos(usuario, contrasena):
    while True:
        try:
            with IMAPClient('mail.utnmail.com', ssl=True) as cliente:
                cliente.login(usuario, contrasena)
                cliente.select_folder('INBOX')
                print(f"Conectado a la cuenta: {usuario}")

                while True:
                    print(f"Esperando nuevos correos para {usuario}...")
                    try:
                        # Espera notificaciones de nuevos correos
                        cliente.idle()
                        respuesta = cliente.idle_check(timeout=300)  # Espera hasta 5 minutos
                        cliente.idle_done()

                        if respuesta:
                            for evento in respuesta:
                                if evento[1] == b'EXISTS':  # Nuevo correo detectado
                                    mensajes_no_leidos = cliente.search(['UNSEEN'])
                                    if mensajes_no_leidos:
                                        mensaje_id = mensajes_no_leidos[-1]
                                        procesar_nuevo_correo(cliente, mensaje_id, usuario)
                    except Exception as e:
                        print(f"Error en IMAP IDLE para {usuario}: {e}")
                        time.sleep(10)
                        break
        except Exception as e:
            print(f"Error general para {usuario}: {e}")
            time.sleep(30)
```

- **Procesamiento Concurrente con Threading:** El procesamiento concurrente fue una característica fundamental del sistema, permitiendo gestionar múltiples cuentas de correo de manera simultánea sin comprometer el rendimiento general. Para implementar esta funcionalidad, la función procesar_correos empleó la biblioteca threading, creando un hilo independiente para cada usuario definido en la lista de credenciales.

  Cada hilo ejecutó la función monitorear_correos, encargada de supervisar continuamente la bandeja de entrada de la cuenta de correo correspondiente. Este enfoque permitió que el sistema gestionara de forma paralela múltiples mensajes provenientes de diferentes cuentas, maximizando la eficiencia operativa.

  Además, esta estrategia resultó especialmente beneficiosa en escenarios con grandes volúmenes de correos electrónicos, ya que garantizó que el programa principal continuara operando sin interrupciones ni bloqueos mientras los eventos asociados a las cuentas eran procesados.

Figura 79

Uso de threading para procesamiento concurrente

```
# Procesar correos para todos los usuarios usando hilos
def procesar_correos():
    hilos = []
    for credenciales in usuarios:
        usuario = credenciales["usuario"]
        contrasena = credenciales["contrasena"]
        hilo = threading.Thread(target=monitorear_correos, args=(usuario, contrasena))
        hilo.start()
        hilos.append(hilo)

    for hilo in hilos:
        hilo.join()
```

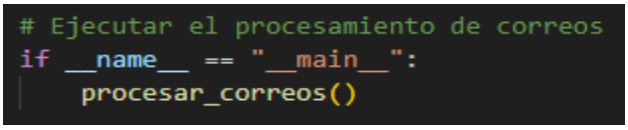
- **Bucle Principal (if __name__ == "__main__":):** El bucle principal garantizó que la función procesar_correos se ejecutara únicamente cuando el script fuese

ejecutado directamente, y no al ser importado como un módulo en otro programa. Este enfoque fue esencial para estructurar adecuadamente el flujo de ejecución del sistema en Python.

Al ejecutarse el script, el sistema inició automáticamente el monitoreo de correos para todos los usuarios definidos en la lista de credenciales. Este bucle actuó como el punto de entrada principal al sistema, activando todos los hilos necesarios para el procesamiento concurrente de los correos electrónicos. Además, aseguró que el sistema operara de manera autónoma, sin requerir intervención manual adicional, proporcionando una solución completamente automatizada para la supervisión y gestión de los mensajes.

Figura 80

Bucle principal para procesar los correos



CAPÍTULO IV. EVALUACIÓN DE RENDIMIENTO

En el cuarto capítulo Evaluación del Rendimiento del Sistema de Detección de Spam, se analizaron métricas importantes como la exactitud, precisión, recall, F1 score y el área bajo la curva ROC (AUC-ROC). Estas métricas proporcionaron una visión clara de cómo los modelos pudieron distinguir entre correos legítimos y spam. Esta evaluación permitió determinar la efectividad general de los modelos y posibles áreas de mejora para optimizar su desempeño.

4.1. Pruebas de Funcionamiento

En este apartado se presentaron las pruebas de funcionamiento realizadas para evaluar el rendimiento de los algoritmos de clasificación KNN, Naive Bayes y Decision Tree en un conjunto de datos. Las pruebas se centraron en los parámetros definidos para cada uno de los modelos y su capacidad para generalizar en datos no vistos. Los modelos fueron evaluados utilizando la curva ROC y el AUC como métricas clave para medir su desempeño en términos de precisión y capacidad de clasificación en un entorno real.

4.1.1. Plan de Pruebas

El plan de pruebas tuvo como objetivo garantizar la calidad y fiabilidad del sistema evaluado mediante la ejecución de diferentes fases de validación, enfocadas en aspectos clave como el envío y recepción de correos electrónicos, la clasificación automática de mensajes, y la generación de métricas comparativas para algoritmos de clasificación. Estas pruebas fueron diseñadas para abordar distintos escenarios y casos de uso, evaluando tanto la funcionalidad del servidor de correos como la eficacia de los algoritmos empleados. A continuación, se detalló la estructura de pruebas, incluyendo los resultados esperados y los criterios para medir el éxito en cada fase.

Tabla 9

Plan de pruebas

ID Prueba	Fase	Descripción	Resultados Esperados
P1	**Pruebas de Envío**	Validar el correcto envío de correos electrónicos individuales y masivos a través del servidor SMTP configurado. Esto incluye verificar que el servidor no genera errores y registra cada correo enviado.	Éxito en la entrega de los correos electrónicos.
P2	**Pruebas de Recepción**	Comprobar que todos los correos enviados al sistema son recibidos exitosamente en la bandeja de entrada del destinatario, sin pérdidas o errores en la transmisión.	Tasa de recepción del 100% de los correos enviados a el sistema.
P3	**Pruebas de Clasificación**	Receptar y clasificar 6000 correos electrónicos utilizando los algoritmos KNN, Naive Bayes y Decision Tree evaluando su capacidad para diferenciar entre correos spam y no spam	Los resultados incluyen tasas de acierto, falsos positivos y negativos, junto con las métricas: Exactitud, Precisión, Recall, F1 Score y AUC-ROC calculadas para los tres algoritmos.

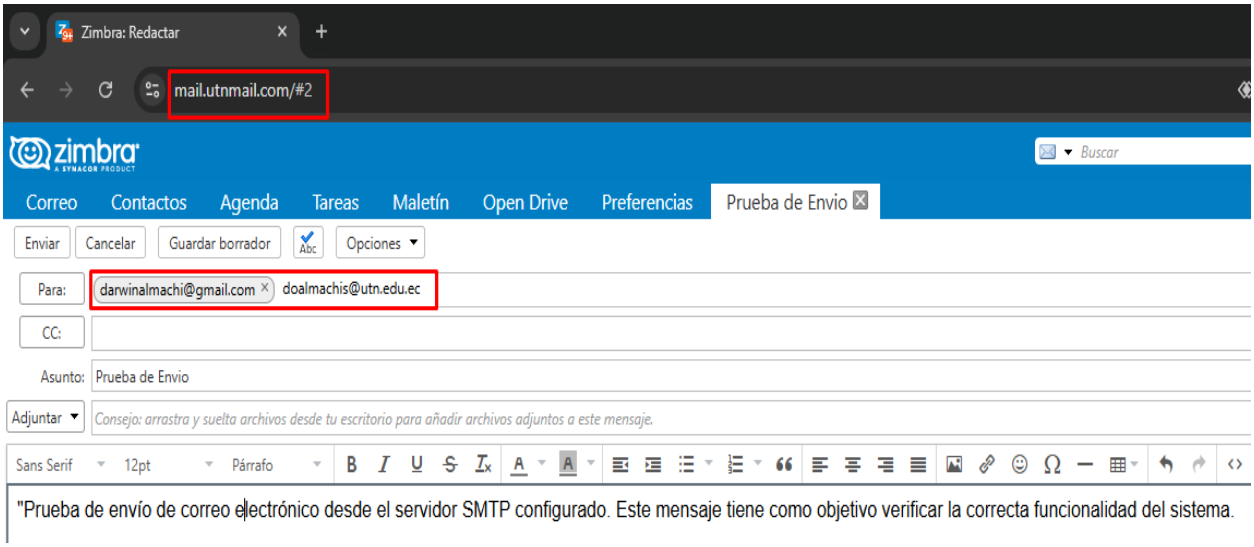
P4	**Evaluación**	Generar y analizar métricas comparativas de desempeño para los tres algoritmos, utilizando los resultados obtenidos en las pruebas de clasificación	Reporte detallado con las métricas clave para Naive Bayes, KNN y Decision Tree, mostrando su eficacia.

4.1.2. Pruebas de Envío de Correos Electrónicos

En este apartado se presentaron las pruebas realizadas para evaluar el funcionamiento y la confiabilidad del envío de correos electrónicos a través del servidor SMTP configurado. Las pruebas incluyeron la correcta redirección de los correos salientes y la funcionalidad de las credenciales de usuario en el servidor SMTP. Además, se probó la compatibilidad del sistema con diferentes clientes de correo.

Figura 81

Envió de correo electrónico desde el servidor de correo Zimbra

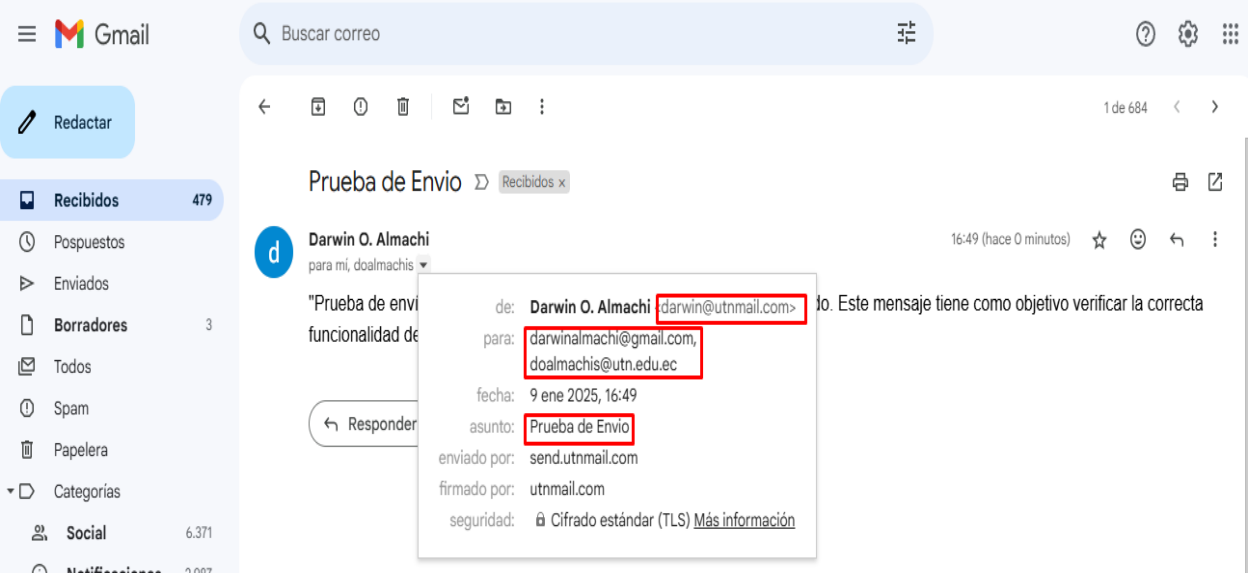


El primer correo electrónico de prueba se envió al servidor de destino Gmail. La configuración del servidor SMTP y las credenciales proporcionadas permitieron un envío

exitoso. La recepción del mensaje fue confirmada sin ningún retraso, como se apreció en la Figura 82, lo que demostró la confiabilidad y eficiencia del sistema al interactuar con uno de los servicios de correo más populares y estrictos en cuanto a seguridad. Este resultado garantizó que los usuarios que utilizaran cuentas de Gmail pudieran recibir comunicaciones sin inconvenientes desde el servidor configurado.

Figura 82

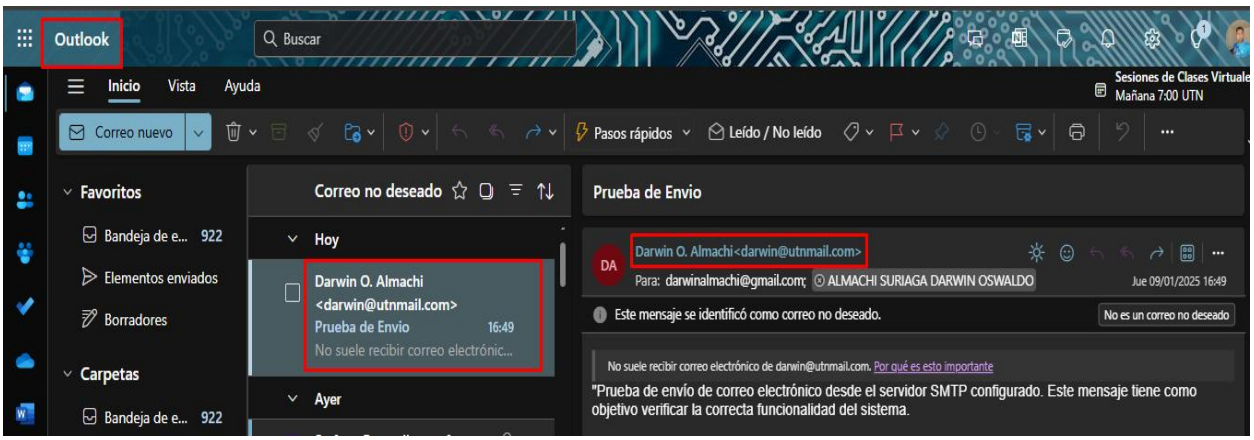
Recepción de correo electrónico enviado en Gmail



La segunda prueba consistió en el envío de un correo electrónico al servidor de destino Outlook. La prueba resultó exitosa, y el mensaje se recibió correctamente en la bandeja de entrada del destinatario, como se observó en la Figura 83. La interacción con el servidor de Microsoft se completó sin errores y con tiempos de respuesta aceptables. Esto indicó que el sistema tenía compatibilidad adecuada con este cliente de correo, asegurando una integración fluida para los usuarios que preferían Outlook como su plataforma de mensajería principal.

Figura 83

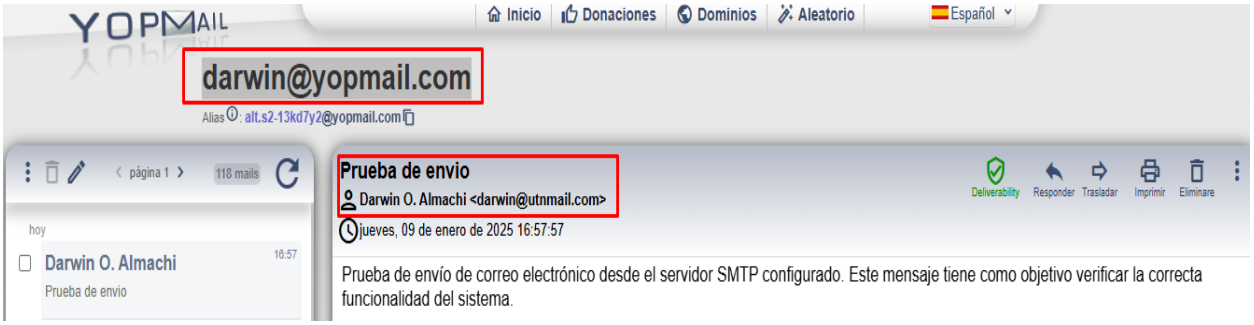
Recepción de correo electrónico enviado en Outlook



Finalmente, se realizó una prueba de envío a Yopmail, un servicio de correo temporal utilizado frecuentemente para verificaciones rápidas. El correo se entregó sin contratiempos, como se visualiza en la Figura 84, demostrando que el sistema podía manejar cuentas de correo no tradicionales o temporales con la misma efectividad que las principales plataformas. Esto reflejó la flexibilidad del servidor SMTP para adaptarse a diversas configuraciones de destinatarios.

Figura 84

Recepción de correo electrónico enviado en Yopmail



Los resultados obtenidos en las pruebas realizadas confirmaron que el sistema cumplía con los estándares necesarios para la entrega de correos electrónicos a distintos servicios de correo, incluidos Gmail, Outlook y Yopmail. En todos los casos, los mensajes fueron entregados

exitosamente, sin retrasos ni errores, y se confirmaron en las bandejas de entrada de los destinatarios correspondientes. Esto aseguró que el servidor SMTP configurado proporcionaba un canal de comunicación seguro, confiable y compatible con múltiples plataformas.

Tabla 10

Tabla de resultados de correos enviados a diferentes servidores de correo

Pruebas de envió de correos electrónicos (P1)			
Servidor de Destino	**Estado del Envío**	**Mensaje Recibido**	**Observaciones**
Gmail	Entregado	Sí	Recibido de inmediato.
Outlook	Entregado	Sí	Compatible sin retrasos.
Yopmail	Entregado	Sí	Recibido correctamente.

4.1.3. Pruebas de Recepción de Correos Electrónicos

En este apartado se describieron las pruebas realizadas para evaluar el correcto funcionamiento y la fiabilidad en la recepción de correos electrónicos. Estas pruebas incluyeron la validación de la configuración del servidor de correo entrante y la capacidad del sistema para manejar diferentes tipos de mensajes. También se evaluó la interoperabilidad con diversos clientes de correo y la respuesta del sistema ante posibles interrupciones en la red.

El sistema fue evaluado para verificar la recepción de correos electrónicos enviados desde cuentas de Gmail hacia el servidor Zimbra configurado. La prueba confirmó que los mensajes enviados desde Gmail se recibieron exitosamente en la bandeja de entrada del servidor. No se observaron demoras significativas ni errores en la transferencia de datos, lo que demostró que la configuración del servidor de correo entrante era robusta y confiable al manejar mensajes provenientes de Gmail.

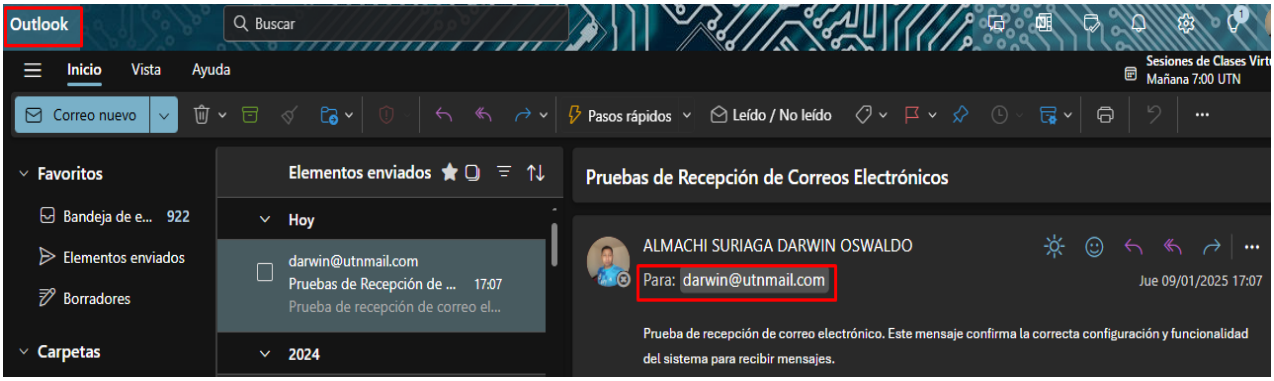
Figura 85

Envió de correo electrónico desde Gmail hacia Zimbra

El segundo experimento consistió en el envío de un correo electrónico desde Outlook al servidor Zimbra. Los resultados fueron igualmente positivos, confirmando que los mensajes se recibieron de forma inmediata y sin pérdida de información. La compatibilidad entre ambas plataformas fue óptima, lo que garantizó que los usuarios de Outlook podían enviar correos al servidor Zimbra sin complicaciones.

Figura 86

Envió de correo electrónico desde Outlook hacia Zimbra



Para evaluar la compatibilidad del sistema con proveedores más especializados como ProtonMail, se realizó una prueba adicional. El mensaje enviado desde ProtonMail llegó al servidor Zimbra de manera eficiente, sin alterar el formato ni la integridad del contenido. Esto

demostró que el servidor podía gestionar mensajes cifrados o provenientes de sistemas con mayores medidas de seguridad.

Figura 87

Envió de correo electrónico desde ProtonMail hacia Zimbra

Se verificó la recepción de los correos electrónicos enviados desde Gmail, Outlook y ProtonMail en el servidor configurado. Los resultados confirmaron que los tres mensajes fueron recibidos exitosamente, como se puede observar en la Figura 88, demostrando la capacidad del sistema para manejar múltiples entradas de distintos servidores de correo simultáneamente. Esto garantizó la interoperabilidad del servidor con una amplia gama de plataformas de correo, proporcionando un canal confiable y eficiente para la recepción de mensajes.

Figura 88

Recepción de correos electrónicos desde los diferentes servidores de correo

Las pruebas realizadas confirmaron que el servidor Zimbra es capaz de recibir correos electrónicos de diferentes proveedores de manera eficiente, segura y sin interrupciones. Esto aseguró que el sistema es interoperable con diversas plataformas, garantizando la integridad y accesibilidad de los mensajes en múltiples escenarios de uso.

Tabla 11

Resultados de correos receptados desde diferentes servidores de correo

Pruebas de recepción de correos electrónicos (P2)			
Servidor de Origen	**Estado de la Recepción**	**Mensaje Recibido**	**Observaciones**
Gmail	Exitoso	Sí	Recibido sin demoras.
Outlook	Exitoso	Sí	Compatible con formato del mensaje.
ProtonMail	Exitoso	Sí	Mensaje intacto y bien formateado.

4.1.4. Pruebas de Clasificación

En este apartado se presentaron las pruebas realizadas para evaluar la efectividad del sistema en la clasificación de correos electrónicos. Estas pruebas incluyeron la implementación y ajuste de algoritmos de clasificación, como KNN, Naive Bayes y Decision Tree, para identificar y categorizar los mensajes según criterios predefinidos. Se emplearon métricas de evaluación como Exactitud, F1 Score, Precision y Recall para medir el rendimiento del sistema.

4.1.4.1. Esquema del funcionamiento

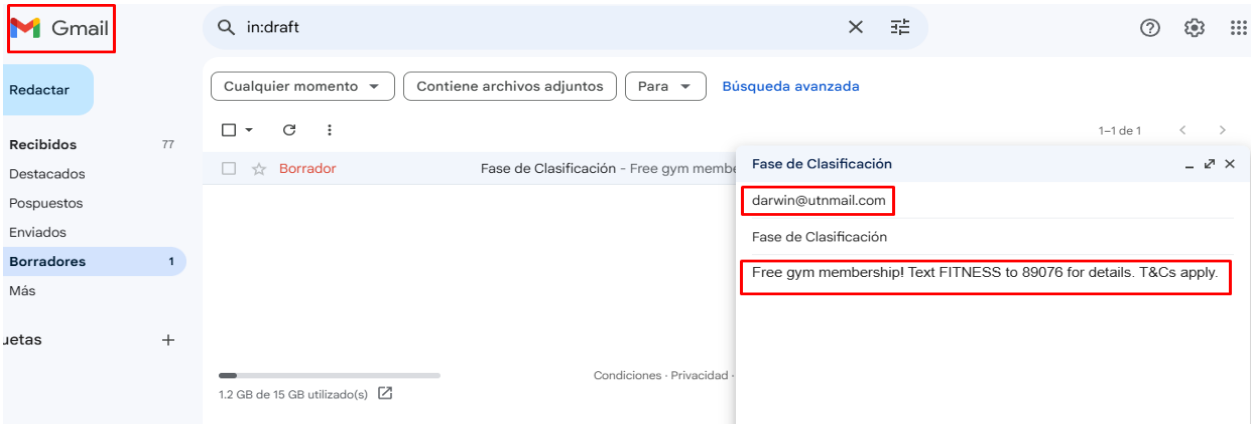
El esquema presentado toma como referencia la Figura 10 de la sección Diagrama de Bloques del Sistema, donde se ilustra el flujo del correo electrónico desde su envío hasta la entrega, pasando por un filtro de spam. El correo se envía desde el dispositivo del remitente a través del protocolo SMTP a su servidor SMTP, que luego lo envía por Internet al servidor SMTP del destinatario. Este lo transfiere al servidor POP/IMAP del destinatario, donde un filtro de spam, mediante un script en Python, analiza el correo para clasificarlo como spam o no spam. Los correos no clasificados como spam se envían a la bandeja de entrada, mientras que los clasificados como spam van a la carpeta de spam, completando el proceso.

- **Envío del Correo Electrónico**

En la Figura 88 se aprecia el proceso inicial en el flujo del correo, donde el mensaje es redactado y enviado desde el dispositivo del remitente a través del protocolo SMTP. Este paso marca el inicio del ciclo de transmisión del mensaje, permitiendo que el servidor SMTP del remitente gestione su envío hacia el destinatario. Es en este punto donde comienza el trayecto del correo a través de Internet, mostrando cómo el sistema integra los elementos básicos de comunicación antes de aplicar los filtros de clasificación.

Figura 89

Envió de correo electrónico desde Gmail

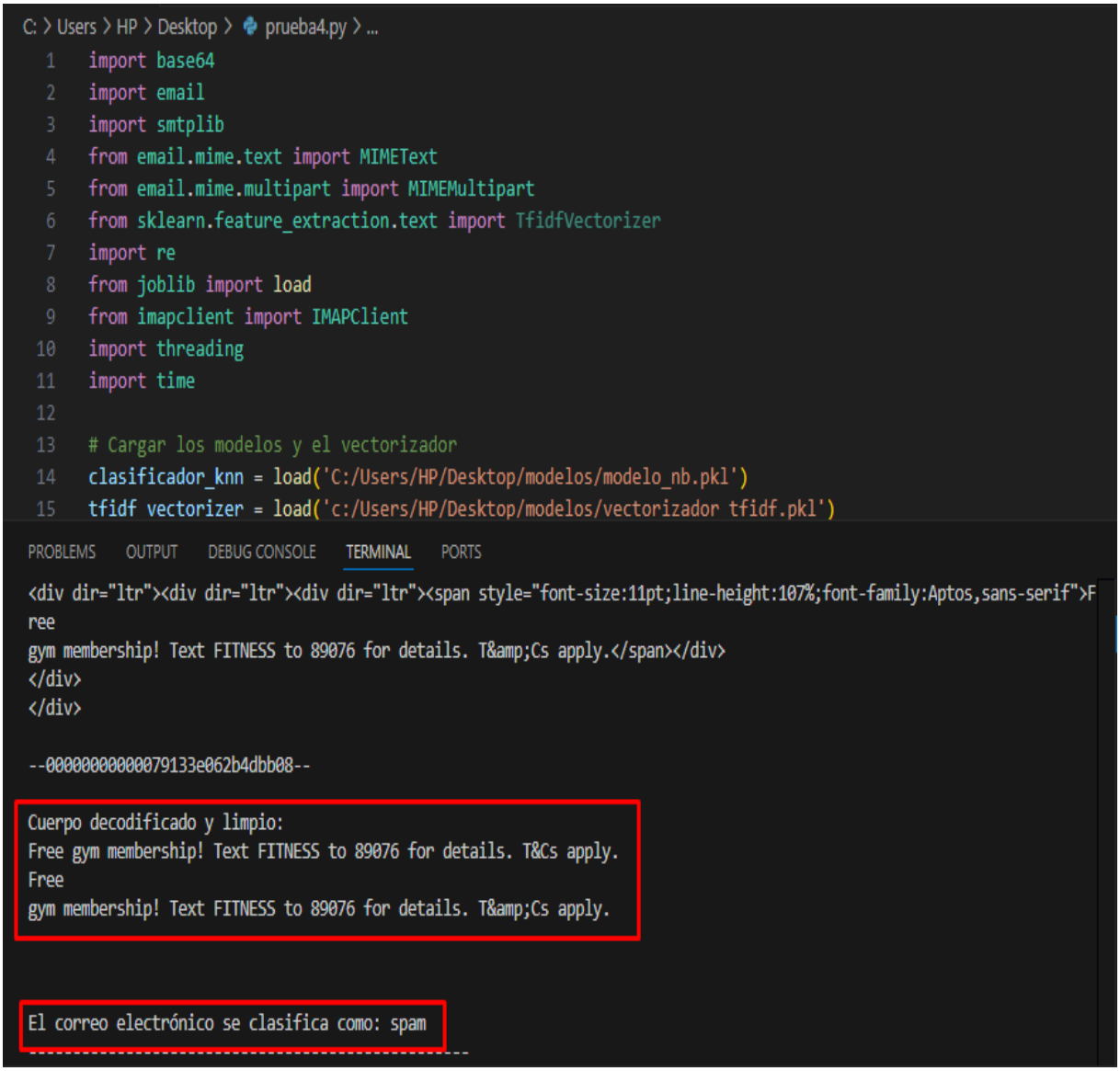


- **Ejecución del Script con el Modelo Clasificador**

La Figura 89, muestra un ejemplo práctico del script de clasificación en funcionamiento, donde un mensaje recibido es analizado mediante el modelo Naive Bayes. El modelo detecta patrones en el contenido del mensaje que lo clasifican como spam. Este paso es esencial para demostrar cómo el sistema logra automatizar la clasificación basándose en las características predefinidas del modelo.

Figura 90

Clasificación de correo entrante con el modelo Naive Bayes

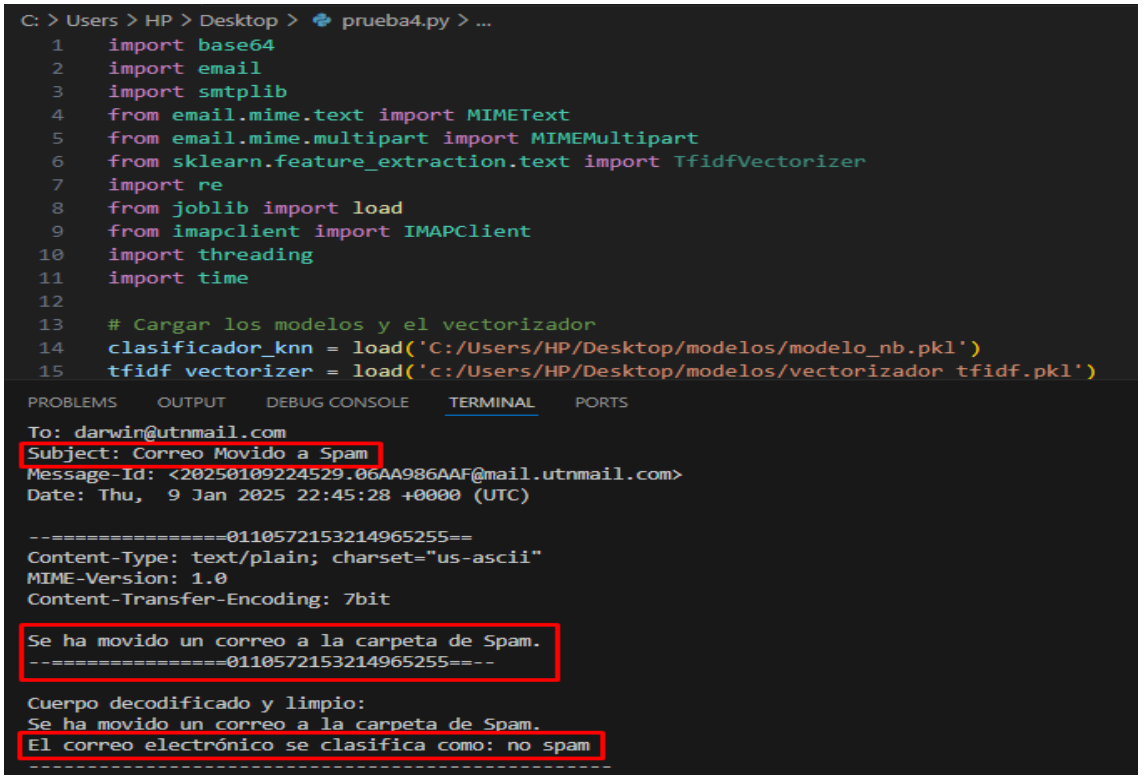


- **Reubicación Automática en la Carpeta de Spam**

En esta etapa, se evidencia el proceso en el cual un mensaje clasificado como spam es reubicado de manera automática en la carpeta destinada para correos no deseados. Este procedimiento asegura que los mensajes no deseados no interfieran con la correcta organización y disponibilidad de los correos legítimos en la bandeja principal del usuario. Además, esta acción refleja la eficiencia del sistema implementado para la clasificación y gestión de mensajes electrónicos, contribuyendo a una experiencia de usuario más ordenada y funcional.

Figura 91

Reubicación de correo catalogado como spam



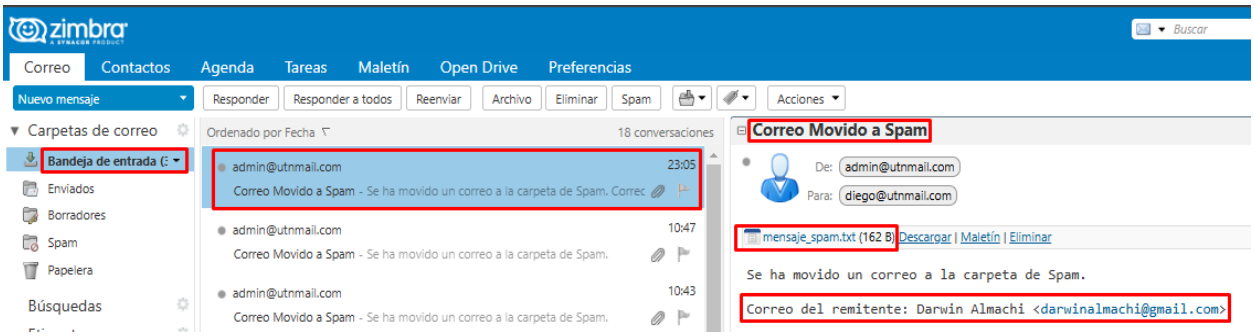
- **Envió de Notificación**

En el panel principal del sistema del servidor de correo, se observa claramente la notificación al usuario, indicando que un mensaje ha sido clasificado como spam y movido automáticamente a la carpeta correspondiente. Este elemento no solo proporciona transparencia

al proceso, sino que también permite al usuario estar informado de manera inmediata sobre las acciones realizadas por el sistema de filtrado. De esta manera, el usuario tiene un control completo sobre la clasificación de los correos electrónicos, lo que facilita la gestión de mensajes y la intervención manual en caso de que sea necesario ajustar o revisar las decisiones del sistema.

Figura 92

Mensaje de alerta sobre correo que ha sido movido a la carpeta de spam

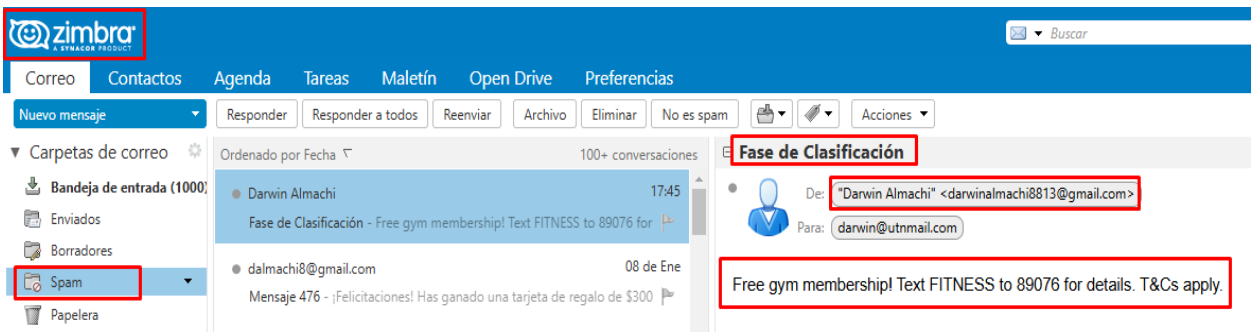


- **Visualización en la Carpeta de Spam**

Se muestra una vista de la carpeta de spam, en la que se almacenan los mensajes clasificados como correos no deseados. Esta representación evidencia que el sistema implementado cumple con su objetivo de organizar y alojar los correos en las ubicaciones asignadas de manera eficiente. Asimismo, garantiza que el usuario pueda acceder a dichos mensajes en caso de ser necesario, asegurando un control y administración adecuados de la información recibida.

Figura 93

Carpeta de spam con el mensaje que el modelo clasifico como spam



4.1.4.2. Algoritmo K-Nearest Neighbors (KNN).

En esta sección se presentaron los resultados de la prueba de clasificación de correos electrónicos utilizando el algoritmo K-Nearest Neighbors (KNN). Para este experimento, se recibieron un total de 2000 mensajes, distribuidos entre correos etiquetados como spam y no spam (ham), con el objetivo de evaluar la capacidad del modelo para distinguir entre ambas clases.

Figura 94

Mensajes receptados de la prueba de clasificación del algoritmo K-Nearest Neighbors (KNN)

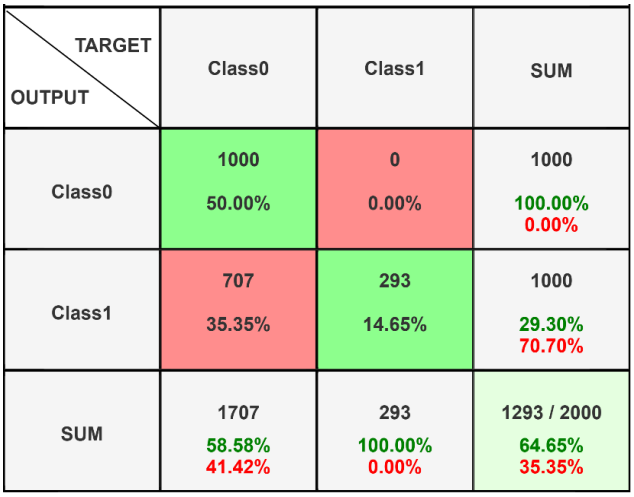
Los resultados obtenidos fueron analizados mediante métricas como la exactitud, precisión, sensibilidad, F1-Score y el área bajo la curva ROC (AUC-ROC), proporcionando una visión integral del desempeño del algoritmo en este escenario de clasificación.

- **Matriz de Confusión**

La matriz detalla la cantidad de predicciones correctas e incorrectas realizadas por el modelo, permitiendo evaluar su precisión, sensibilidad y especificidad. Este análisis es crucial para identificar posibles áreas de mejora en el rendimiento del clasificador y garantizar su eficacia en la separación de correos deseados y no deseados en escenarios prácticos.

Figura 95

Matriz de confusión de la prueba de clasificación del algoritmo K-Nearest Neighbors (KNN)



La Figura 95 presenta una matriz de confusión que ilustra el desempeño del modelo de clasificación K-Nearest Neighbors (KNN) en un entorno de funcionamiento real. Los resultados están organizados en dos categorías: Clase 0 (Ham), que representa los correos legítimos, y Clase 1 (Spam), correspondiente a los correos no deseados.

Tabla 12

Resumen de desempeño de los resultados del modelo K-Nearest Neighbors

Pruebas de clasificación de correos electrónicos (P3)		
Clase	**Métrica**	**Cantidad**
Spam (Clase 1)	Verdaderos Positivos (TP)	293

Ham (Clase 0)	Falsos Negativos (FN)	707
	Verdaderos Negativos (TN)	1000
	Falsos Positivos (FP)	0
Total		2000

- **Cálculo de Métricas de Rendimiento de Clasificación**

A continuación, se calcularon las métricas de rendimiento obtenidas al aplicar el algoritmo K-Nearest Neighbors (KNN) en la clasificación de correos electrónicos. Estas métricas permitieron evaluar la efectividad del modelo al clasificar correctamente mensajes como spam o ham, proporcionando un análisis detallado del desempeño. Los cálculos presentados permitieron interpretar los puntos fuertes y las limitaciones del modelo en este contexto experimental.

- **Exactitud (Accuracy):**

La exactitud se calcula utilizando la Ecuación (8):

$$\textbf{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

Reemplazando los valores en la ecuación:

$$\textbf{Exactitud} = \frac{293 + 1000}{293 + 1000 + 0 + 707} = 0.6465 \qquad (13)$$

- **Precisión (Clase Spam):**

La precisión se calcula utilizando la Ecuación (9):

$$\textbf{Precisión} = \frac{VP}{VP + FP}$$

Sustituyendo los valores en la ecuación:

$$\textbf{Precisión} = \frac{293}{293 + 0} = 1.00 \quad (14)$$

- **Recall (Sensibilidad, Clase Spam):**

  El Recall se calcula utilizando la Ecuación (10):

$$\textbf{Recall} = \frac{VP}{VP + FN}$$

  Insertando los valores en la ecuación:

$$\textbf{Recall} = \frac{293}{293 + 707} = 0.293 \quad (15)$$

- **F1-Score:**

  El F1-Score se calcula utilizando la Ecuación (11):

$$\textbf{F1 Score} = 2 \times \frac{Precisión \times Recall}{Precisión + Recall}$$

  Reemplazando los valores en la ecuación:

$$\textbf{F1 Score} = 2 \times \frac{1.00 \times 0.293}{1.00 + 0.293} = 0.453 \quad (16)$$

- **AUC-ROC:**

  Especificidad

$$\textbf{Especificidad} = \frac{VN}{VN + FP} = \frac{1000}{1000 + 0} = 1.00 \quad (17)$$

  El AUC-ROC se calcula utilizando la Ecuación (12):

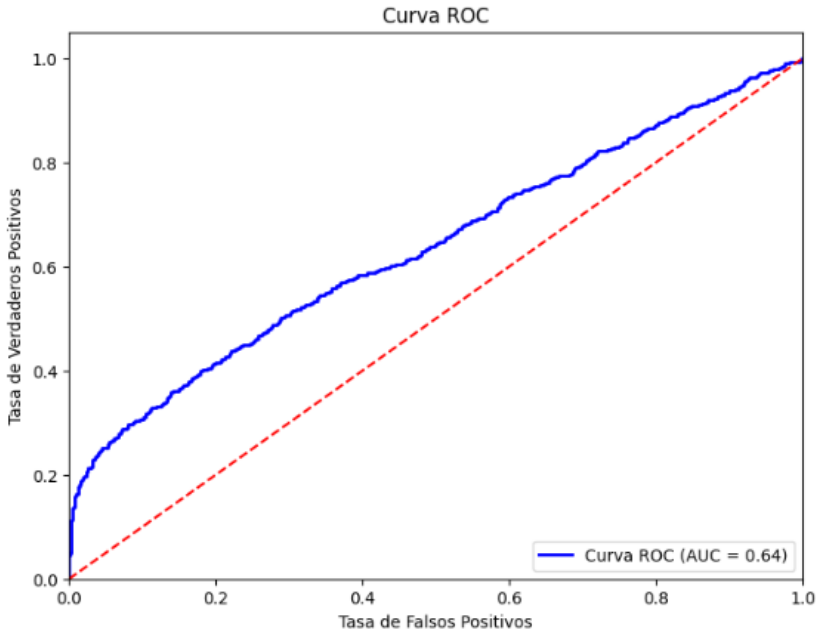
$$\textbf{AUC} - \textbf{ROC} = \frac{Recall + Especifiicidad}{2}$$

Sustituyendo los valores en la ecuación:

$$\mathbf{AUC - ROC} = \frac{0.293 + 1.0}{2} = 0.6465 \qquad (\ 18\ )$$

Figura 96

La Curva ROC muestra la relación entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR) para el modelo K-Nearest Neighbors



La Figura 96 de la curva ROC, muestra la relación entre la Tasa de Verdaderos Positivos (TPR) y la Tasa de Falsos Positivos (FPR), con un AUC de 0.64 que refleja la capacidad del modelo K-Nearest Neighbors para distinguir entre las clases (spam y ham).

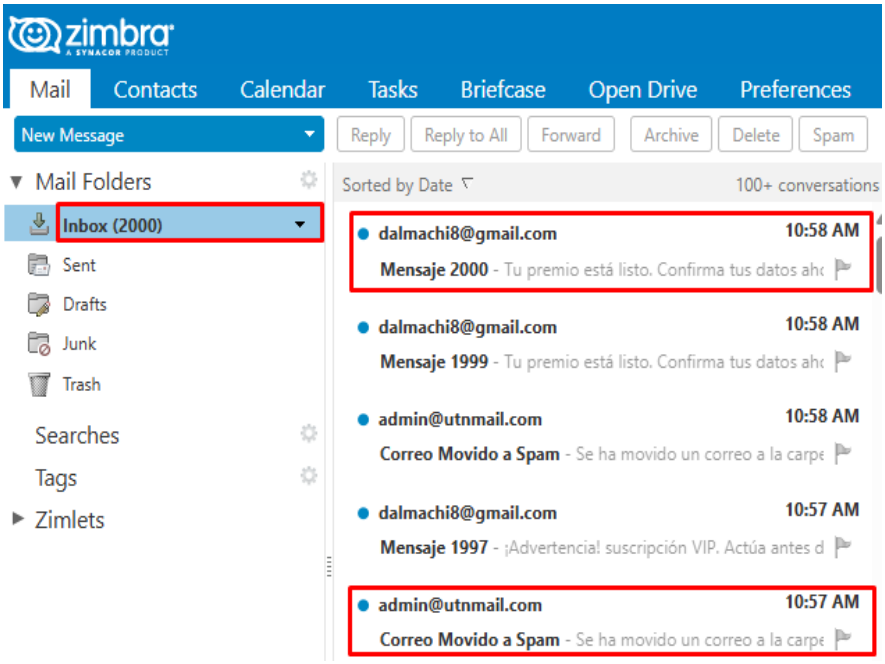
4.1.4.3. Algoritmo Naive Bayes

El algoritmo Naive Bayes fue aplicado para la clasificación de correos electrónicos, considerando un total de 2000 mensajes que incluían tanto spam como ham. Este método, conocido por su rapidez y eficiencia en problemas de clasificación, se basaba en principios probabilísticos que asumían la independencia de las características utilizadas. Los resultados se

analizaron a partir de una matriz de confusión, que permitió observar el comportamiento del modelo en términos de aciertos y errores.

Figura 97

Mensajes receptados de la prueba de clasificación del algoritmo Naive Bayes



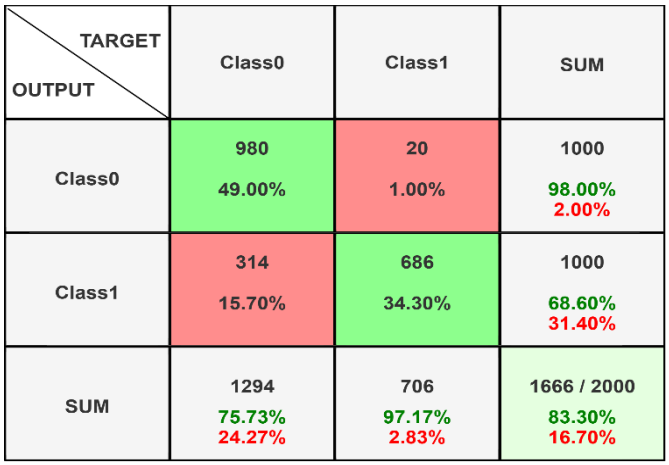
Además, se calcularon métricas clave como la exactitud, la sensibilidad y el F1-Score, con el objetivo de evaluar la capacidad del modelo para identificar correctamente correos pertenecientes a ambas clases.

- **Matriz de Confusión**

La matriz refleja la cantidad de predicciones acertadas y fallidas realizadas por el modelo, lo que permite analizar indicadores clave como la precisión, la sensibilidad y la especificidad. Este estudio es esencial para identificar posibles mejoras en el funcionamiento del clasificador Naive Bayes y garantizar su capacidad para diferenciar correctamente entre correos legítimos y correos no deseados en aplicaciones reales.

Figura 98

Matriz de confusión de la prueba de clasificación del algoritmo Naive Bayes



La Figura 98 presenta una matriz de confusión que ilustró el desempeño del modelo de clasificación Naive Bayes en un escenario de aplicación real. En esta matriz se analizaron las predicciones realizadas por el modelo para las dos categorías principales: Clase 0 (Ham) y Clase 1 (Spam). Los datos reflejaron tanto las decisiones correctas del clasificador como los errores cometidos, permitiendo evaluar su capacidad para diferenciar entre mensajes spam y no spam.

Tabla 13

Resumen de desempeño de los resultados del modelo Naive Bayes

Pruebas de clasificación de correos electrónicos (P3)		
Clase	Métrica	Cantidad
Spam (Clase 1)	Verdaderos Positivos (VP)	686
	Falsos Negativos (FN)	314
Ham (Clase 0)	Verdaderos Negativos (VN)	980
	Falsos Positivos (FP)	20
Total		2000

- **Cálculo de Métricas de Rendimiento de Clasificación**

Se presentaron las métricas de rendimiento correspondientes al algoritmo Naive Bayes, utilizadas para evaluar su desempeño en la tarea de clasificación de correos electrónicos. Estas métricas permitieron medir la capacidad del modelo para identificar correctamente mensajes spam y ham, ofreciendo un análisis detallado de su efectividad y las áreas en las que podría mejorar en un entorno de aplicación real.

- **Exactitud (Accuracy):**

La exactitud se calcula utilizando la Ecuación (8):

$$\textbf{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

Reemplazando los valores en la ecuación:

$$\textbf{Exactitud} = \frac{686 + 980}{686 + 980 + 20 + 314} = 0.833 \qquad (19)$$

- **Precisión (Clase Spam):**

La precisión se calcula utilizando la Ecuación (9):

$$\textbf{Precisión} = \frac{VP}{VP + FP}$$

Sustituyendo los valores en la ecuación:

$$\textbf{Precisión} = \frac{686}{686 + 20} = 0.9717 \qquad (20)$$

Recall (Sensibilidad, Clase Spam):

El Recall se calcula utilizando la Ecuación (10):

$$\textbf{Recall} = \frac{VP}{VP + FN}$$

Insertando los valores en la ecuación:

$$\textbf{Recall} = \frac{686}{686 + 314} = 0.686 \qquad (21)$$

F1-Score:

El F1 Score se calcula utilizando la Ecuación (11):

$$\textbf{F1 Score} = 2 \times \frac{Precisión \times Recall}{Precisión + Recall}$$

Reemplazando los valores en la ecuación:

$$\textbf{F1 Score} = 2 \times \frac{0.9717 \times 0.686}{0.9717 + 0.686} = 0.804 \qquad (22)$$

- **AUC-ROC:**

Especificidad

$$\textbf{Especificidad} = \frac{VN}{VN + FP} = \frac{980}{980 + 20} = 0.98 \qquad (23)$$

El AUC-ROC se calcula utilizando la Ecuación (12):

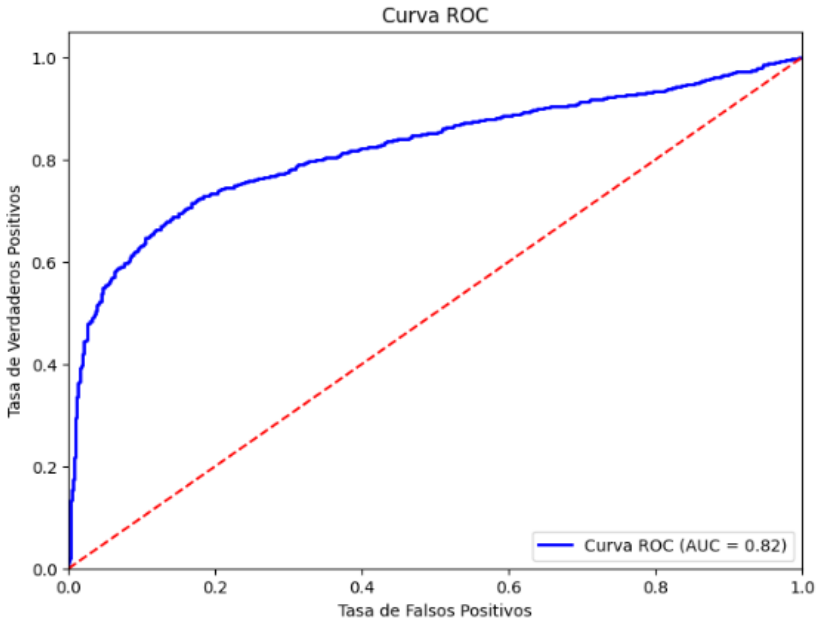
$$\textbf{AUC} - \textbf{ROC} = \frac{Recall + Especifiicidad}{2}$$

Sustituyendo los valores en la ecuación:

$$\textbf{AUC} - \textbf{ROC} = \frac{0.686 + 0.98}{2} = 0.833 \qquad (24)$$

Figura 99

La Curva ROC muestra la relación entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR) del modelo Naive Bayes



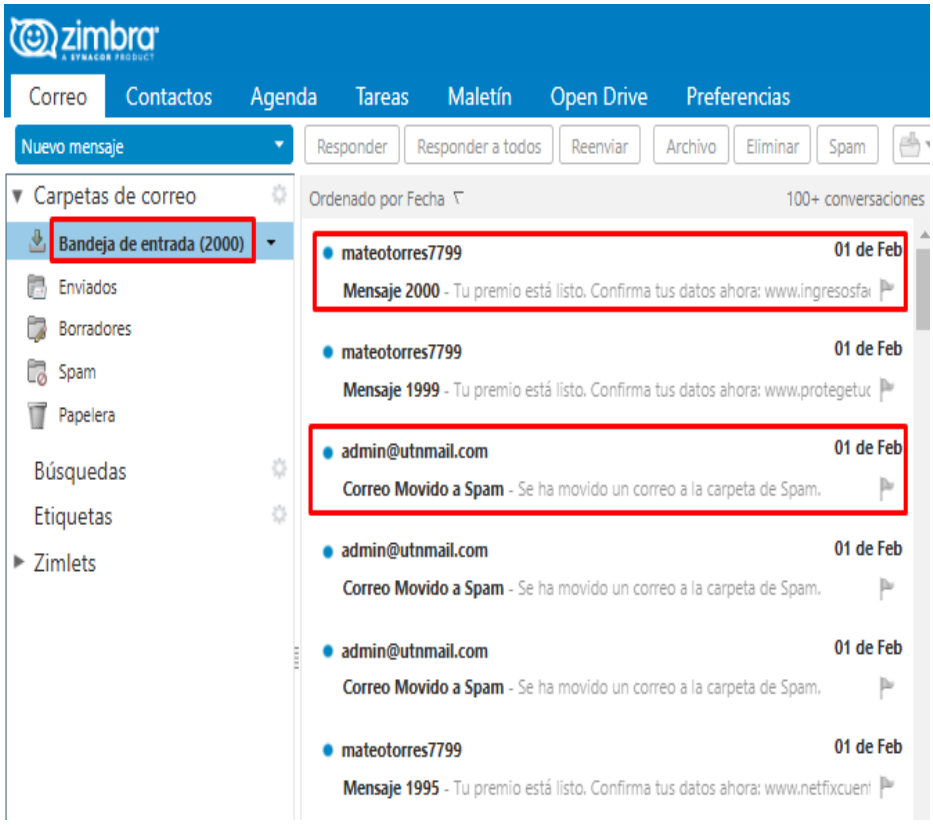
La Figura 99 de la curva ROC refleja la relación entre la Tasa de Verdaderos Positivos (TPR) y la Tasa de Falsos Positivos (FPR). Con un AUC de 0.82, se destaca la efectividad del modelo Naive Bayes para discriminar con precisión entre las categorías de spam y ham, demostrando su capacidad para abordar esta tarea de clasificación de manera confiable.

4.1.4.4. Algoritmo Decision Tree

El modelo Decision Tree se empleó para llevar a cabo la clasificación de un conjunto de 2000 correos electrónicos, distribuidos equitativamente entre las categorías de spam y ham. Este enfoque se basó en la construcción de un árbol de decisiones que segmentaba los datos en función de atributos relevantes, optimizando la separación entre clases. A través de una matriz de confusión, se evaluó el desempeño del modelo, analizando tanto los aciertos como los errores en la clasificación.

Figura 100

Mensajes receptados de la prueba de clasificación del algoritmo Decision Tree



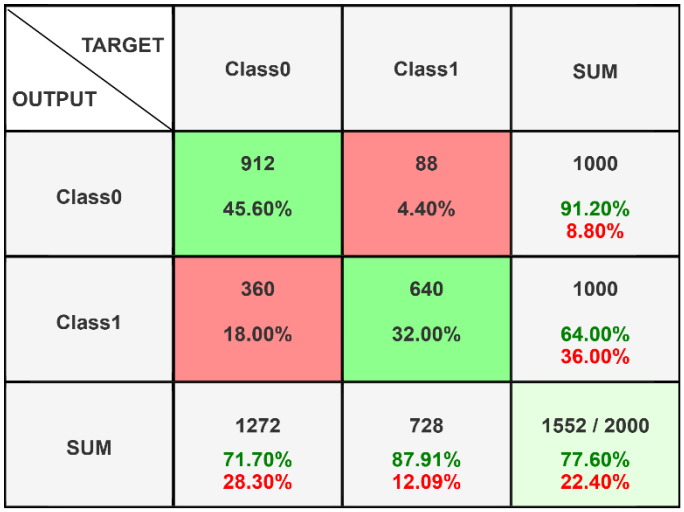
Asimismo, se calcularon indicadores como la precisión, el recall y el F1-Score, con el propósito de determinar la efectividad del algoritmo en la detección de correos de ambas categorías y su capacidad para minimizar errores de clasificación.

- **Matriz de Confusión**

La matriz representa el número de predicciones correctas e incorrectas realizadas por el modelo, proporcionando información valiosa para evaluar métricas como precisión, sensibilidad y especificidad. Este análisis es clave para detectar áreas de mejora en el rendimiento del clasificador Decision Tree y asegurar su eficacia en la clasificación de correos legítimos y no deseados en entornos reales.

Figura 101

Matriz de confusión de la prueba de clasificación del algoritmo Decision Tree



La Figura 101 muestra una matriz de confusión que evaluó el desempeño del modelo de clasificación Decision Tree en un entorno práctico. En esta representación se analizaron las predicciones del modelo para las categorías principales: Clase 0 (Ham) y Clase 1 (Spam). La matriz destacó tanto los aciertos como los errores del clasificador, proporcionando una visión clara de su capacidad para identificar correctamente cada tipo de mensaje y de los patrones donde presentó limitaciones, lo cual fue clave para optimizar su efectividad en tareas de clasificación.

Tabla 14

Resumen de desempeño de los resultados del modelo Decision Tree

Pruebas de clasificación de correos electrónicos (P3)		
Clase	Métrica	Cantidad
Spam (Clase 1)	Verdaderos Positivos (VP)	640
	Falsos Negativos (FN)	360
Ham (Clase 0)	Verdaderos Negativos (VN)	912

	Falsos Positivos (FP)	88
Total		2000

- **Cálculo de Métricas de Rendimiento de Clasificación**

El algoritmo Decision Tree fue evaluado a través de diversas métricas de rendimiento, las cuales proporcionaron una visión integral sobre su capacidad para clasificar correctamente los correos electrónicos en las categorías de spam y ham. Estas métricas permitieron analizar tanto la precisión global del modelo como su habilidad para manejar desequilibrios en los datos y capturar patrones relevantes. Además, ofrecieron información valiosa sobre posibles ajustes en la estructura del árbol para optimizar su desempeño en aplicaciones prácticas y maximizar su efectividad en contextos reales.

- **Exactitud (Accuracy):**

La exactitud se calcula utilizando la Ecuación (8):

$$\textbf{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}$$

Reemplazando los valores en la ecuación:

$$\textbf{Exactitud} = \frac{640 + 912}{640 + 912 + 88 + 360} = 0.776 \qquad (25)$$

- **Precisión (Clase Spam):**

La precisión se calcula utilizando la Ecuación (9):

$$\textbf{Precisión} = \frac{VP}{VP + FP}$$

Sustituyendo los valores en la ecuación:

$$\textbf{Precisión} = \frac{640}{640 + 88} = 0.8791 \quad (26)$$

- **Recall (Sensibilidad, Clase Spam):**

El Recall se calcula utilizando la Ecuación (10):

$$\textbf{Recall} = \frac{VP}{VP + FN}$$

Insertando los valores en la ecuación:

$$\textbf{Recall} = \frac{640}{640 + 360} = 0.64 \quad (27)$$

- **F1-Score:**

El F1 Score se calcula utilizando la Ecuación (11):

$$\textbf{F1 Score} = 2 \times \frac{Precisión \times Recall}{Precisión + Recall}$$

Reemplazando los valores en la ecuación:

$$\textbf{F1 Score} = 2 \times \frac{0.8791 \times 0.64}{0.8791 + 0.64} \approx 0.7406 \quad (28)$$

- **AUC-ROC:**

Especificidad

$$\textbf{Especificidad} = \frac{VN}{VN + FP} = \frac{912}{912 + 88} = 0.91 \quad (29)$$

El AUC-ROC se calcula utilizando la Ecuación (12):

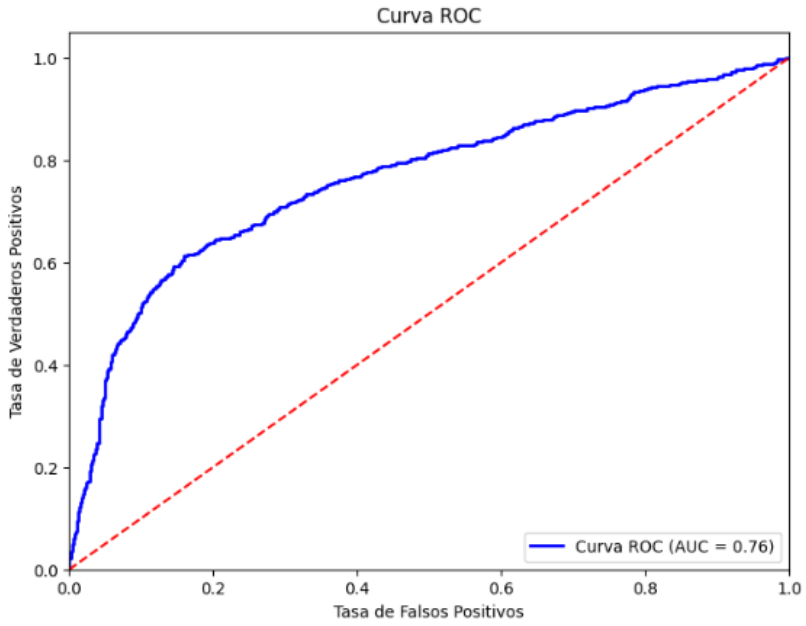
$$\textbf{AUC} - \textbf{ROC} = \frac{Recall + Especifiicidad}{2}$$

Sustituyendo los valores en la ecuación:

$$\mathbf{AUC - ROC} = \frac{0.64 + 0.91}{2} = 0.775 \qquad (\ 30\ )$$

Figura 102

La Curva ROC muestra la relación entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR) para el modelo Decision Tree



La Figura 102 de la curva ROC ilustra la conexión entre la Tasa de Verdaderos Positivos (TPR) y la Tasa de Falsos Positivos (FPR). Con un AUC de 0.76, se evidenció la eficacia del modelo Decision Tree en diferenciar correctamente entre las dos categorías: spam y ham.

4.1.5. Evaluación

La evaluación de los algoritmos Naive Bayes, K-Nearest Neighbors (KNN) y Decision Tree se llevó a cabo considerando dos escenarios principales: las métricas obtenidas en las fases de entrenamiento y prueba, y los resultados en un entorno real de clasificación de correos electrónicos. Los análisis realizados permitieron observar las fortalezas y debilidades de cada

algoritmo en distintas condiciones, ofreciendo una evaluación integral para la selección del modelo más adecuado en el contexto de detección de spam.

4.1.5.1. Rendimiento en Entrenamiento y Validación

Los resultados de las métricas evaluadas en las fases de entrenamiento y prueba permitieron identificar el comportamiento de cada algoritmo en un entorno controlado. En este contexto, se destacó que Naive Bayes presentó un rendimiento consistente con una precisión alta tanto en el conjunto de entrenamiento como en el de prueba. Sin embargo, se observó una disminución en el recall durante la fase de prueba, lo que indicó que el modelo podría no ser tan efectivo en identificar correos spam en situaciones prácticas.

Por otro lado, el algoritmo KNN alcanzó una precisión perfecta en la fase de prueba, pero su bajo recall limitó su efectividad en la detección de spam, lo que resultó en un F1-Score reducido. Finalmente, el modelo Decision Tree mostró un rendimiento equilibrado entre precisión y recall, destacándose por su robustez en la clasificación de correos en ambas fases. Los detalles de las métricas obtenidas se presentaron en la Tabla 15.

Tabla 15

Comparación de métricas de rendimiento en las fases de entrenamiento y prueba para Naive Bayes, KNN y Decision Tree.

Algoritmo	Conjunto	Exactitud	Precisión	Recall	F1-Score	AUC-ROC
Naive Bayes	Entrenamiento	0.99	1	1	1	1
Naive Bayes	Validación	0.98	0.99	0.99	0.95	0.89

KNN	Entrenamiento	0.94	0.93	1	0.74	0.98
KNN	Validación	0.93	0.94	0.58	0.73	0.71
Decision Tree	Entrenamiento	0.99	0.99	1	1	0.93
Decision Tree	Validación	0.96	0.92	0.84	0.88	0.80

4.1.5.2. Resultados en un Entorno Real

En el escenario práctico, se evaluó el desempeño de los algoritmos en la clasificación de 2000 correos electrónicos, distribuidos entre mensajes spam y no spam. Los resultados obtenidos reflejaron que Naive Bayes se destacó por su precisión elevada y un buen balance entre precisión y recall, lo que resultó en un F1-Score sobresaliente. En contraste, aunque KNN presentó una precisión perfecta al identificar correos no spam, su bajo recall redujo significativamente su capacidad para detectar mensajes spam, limitando su efectividad global. Por otro lado, el modelo Decision Tree mostró un desempeño intermedio, logrando un equilibrio aceptable entre precisión y recall. Los datos detallados se presentaron en la Tabla 16.

Tabla 16

Resultados de las métricas de rendimiento en un entorno real para Naive Bayes, KNN y Decision Tree.

Algoritmo	VP	VN	FP	FN	Exactitud	Precisión	Recall	F1-Score	AUC-ROC
Naive Bayes	686	980	20	314	0.833	0.9717	0.686	0.804	0.833
KNN	293	1000	0	707	0.6465	1	0.293	0.453	0.6465
Decision Tree	640	912	88	360	0.776	0.8791	0.64	0.74	0.776

Nota. VP (Verdaderos Positivos) se refiere a los casos correctamente clasificados como positivos, VN (Verdaderos Negativos) a los casos correctamente clasificados como negativos, FP (Falsos

Positivos) a los casos incorrectamente clasificados como positivos y FN (Falsos Negativos) a los casos incorrectamente clasificados como negativos.

4.1.5.3. Análisis Comparativo

Se evaluó el desempeño de los algoritmos Naive Bayes, KNN y Decision Tree en diferentes escenarios, con el objetivo de determinar cuál ofrece el mejor rendimiento en términos de precisión, sensibilidad (recall) y eficiencia en la detección de spam. Los resultados, detallados en las Tablas 15 y 16, permitieron extraer las siguientes observaciones.

- **Naive Bayes**

Naive Bayes mostró un desempeño consistente en los escenarios evaluados. En las fases de entrenamiento y validación, obtuvo valores altos de precisión y exactitud, destacándose como el modelo más confiable para generalizar a nuevos datos. Sin embargo, su leve disminución en el recall (0.686 en entorno real) indica que podría tener dificultades para detectar correos spam en casos más complejos o personalizados. A pesar de ello, su equilibrio entre precisión y recall lo posiciona como el modelo con el F1-Score más alto en entorno real (0.804), consolidándose como la opción más robusta para la clasificación de correos electrónicos no deseados.

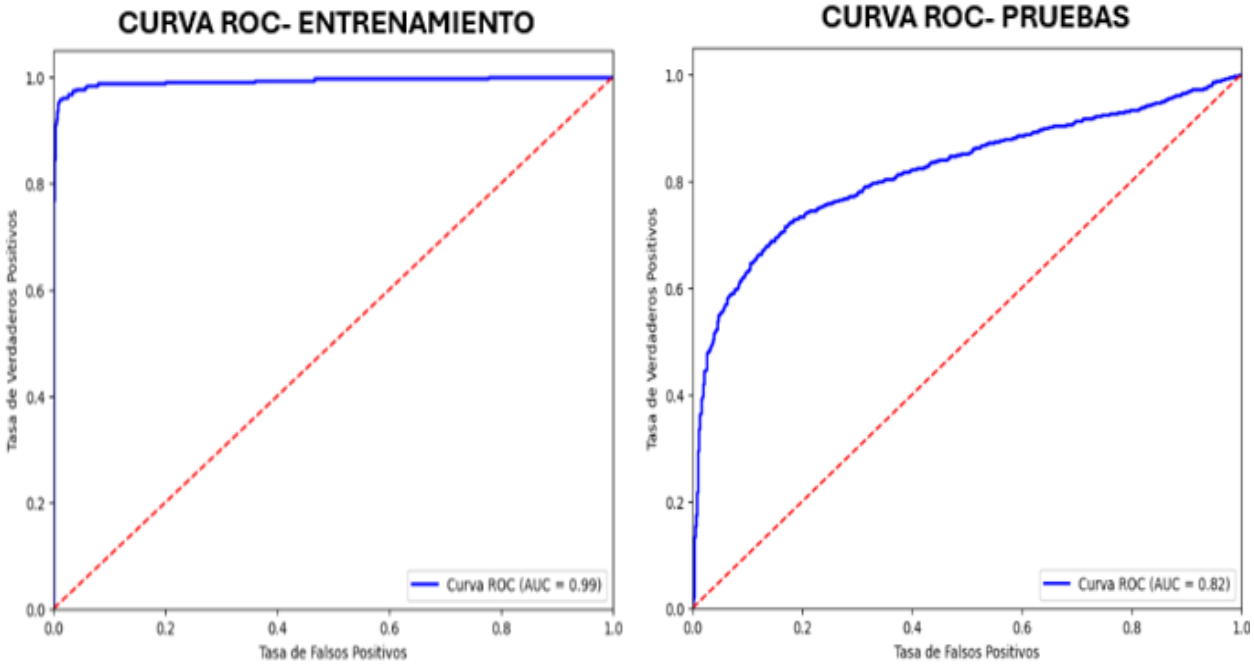
Esta tendencia se refleja en la Figura 103, que muestra las curvas ROC del modelo en las fases de entrenamiento y pruebas en un entorno real. En la fase de entrenamiento, el área bajo la curva (AUC = 0.99) evidencia un rendimiento casi perfecto, lo que sugiere que el modelo distingue de manera muy efectiva entre correos spam y no spam en datos previamente vistos. Sin embargo, en la fase de pruebas reales, el AUC disminuye a 0.82, lo que indica una reducción en la capacidad del modelo para mantener este nivel de discriminación cuando se enfrenta a datos nuevos y no estructurados de la misma manera.

Esta diferencia se debe, en gran parte, a la naturaleza más compleja y diversa de los datos del entorno real, donde pueden presentarse patrones no observados durante el entrenamiento. Factores como variaciones en el lenguaje, estructuras de los correos, y tácticas nuevas de spam pueden afectar la capacidad del modelo para identificar correctamente los correos no deseados. Además, el modelo tiende a ser más optimista durante el entrenamiento, ya que está adaptado a los datos sobre los que ha aprendido, mientras que en pruebas reales enfrenta escenarios menos predecibles.

A pesar de esta disminución en el AUC, el rendimiento sigue siendo sólido, lo que confirma la capacidad del modelo para generalizar de manera efectiva. La comparación de ambas curvas permite identificar claramente el desafío de mantener un equilibrio entre la sensibilidad y la especificidad en condiciones del mundo real, subrayando la importancia de validar modelos en entornos diversos para asegurar su robustez.

Figura 103

Curvas ROC del Modelo Naive Bayes en fase de entrenamiento y pruebas



- **K-Nearest Neighbor (KNN)**

KNN presentó un comportamiento contrastante: logró una precisión perfecta (1.00) en el entorno real al identificar correos no spam, pero su bajo recall (0.293) limitó significativamente su capacidad para detectar mensajes spam. Esto puede atribuirse a la sensibilidad del modelo a desequilibrios en los datos y a la selección del valor de 'k'. Aunque es un algoritmo eficaz en escenarios controlados, su desempeño en aplicaciones prácticas lo hace menos adecuado para la tarea de detección de spam.

Esta tendencia se refleja en la Figura 104, que muestra las curvas ROC del modelo en las fases de entrenamiento y pruebas en un entorno real. En la fase de entrenamiento, el área bajo la curva (AUC = 0.98) evidencia un rendimiento casi perfecto, lo que sugiere que el modelo distingue de manera efectiva entre correos spam y no spam en datos previamente vistos. Sin embargo, en la fase de pruebas reales, el AUC disminuye a 0.64, lo que indica una reducción notable en la capacidad del modelo para mantener este nivel de discriminación cuando se enfrenta a datos nuevos y más variados.

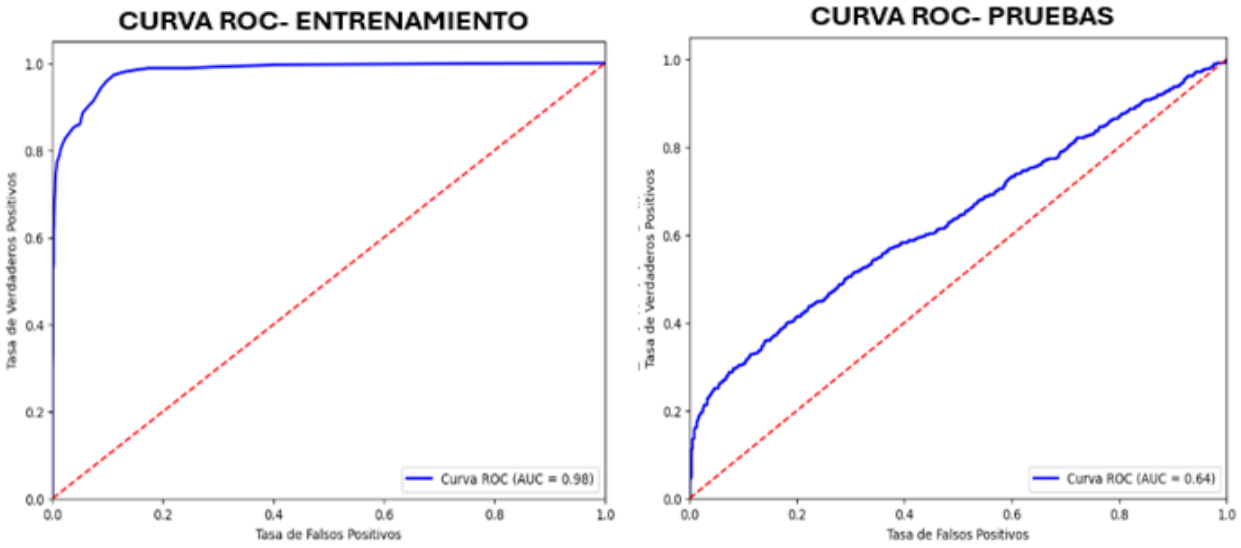
Esta diferencia se debe a la dependencia de KNN en la distribución de los datos. Durante el entrenamiento, el modelo tiene acceso a datos estructurados con patrones bien definidos, lo que permite una clasificación precisa. Sin embargo, en un entorno real, la estructura de los correos electrónicos varía considerablemente, dificultando la identificación de mensajes spam. Además, la selección del valor de 'k' juega un papel fundamental, ya que valores inadecuados pueden hacer que el modelo sea demasiado sensible a la variabilidad de los datos o que pierda capacidad de detección.

A pesar de esta disminución en el AUC, la comparación de ambas curvas deja en evidencia la dificultad de KNN para generalizar eficazmente en condiciones del mundo real.

Mientras que en el entrenamiento logra un desempeño ideal, en la práctica muestra un sesgo significativo hacia la clasificación de correos como no spam, afectando su utilidad en la detección de mensajes no deseados. Esto resalta la importancia de validar modelos en escenarios diversos para asegurar su robustez antes de su implementación definitiva.

Figura 104

Curvas ROC del Modelo KNN en fase de entrenamiento y pruebas



- **Decision Tree**

Decision Tree mostró un rendimiento equilibrado en todos los escenarios evaluados, destacándose por su robustez. Su precisión y recall moderados (0.8791 y 0.64 en entorno real, respectivamente) resultaron en un F1-Score de 0.74, ubicándolo como una alternativa intermedia entre Naive Bayes y KNN. Aunque su AUC-ROC de 0.76 evidencia una capacidad razonable para diferenciar entre clases, este modelo podría beneficiarse de técnicas adicionales de ajuste para maximizar su desempeño.

Esta tendencia se refleja en la Figura 105, que muestra las curvas ROC del modelo en las fases de entrenamiento y pruebas en un entorno real. En la fase de entrenamiento, el área bajo la curva (AUC = 0.93) indica un rendimiento sólido, lo que sugiere que el modelo logra distinguir

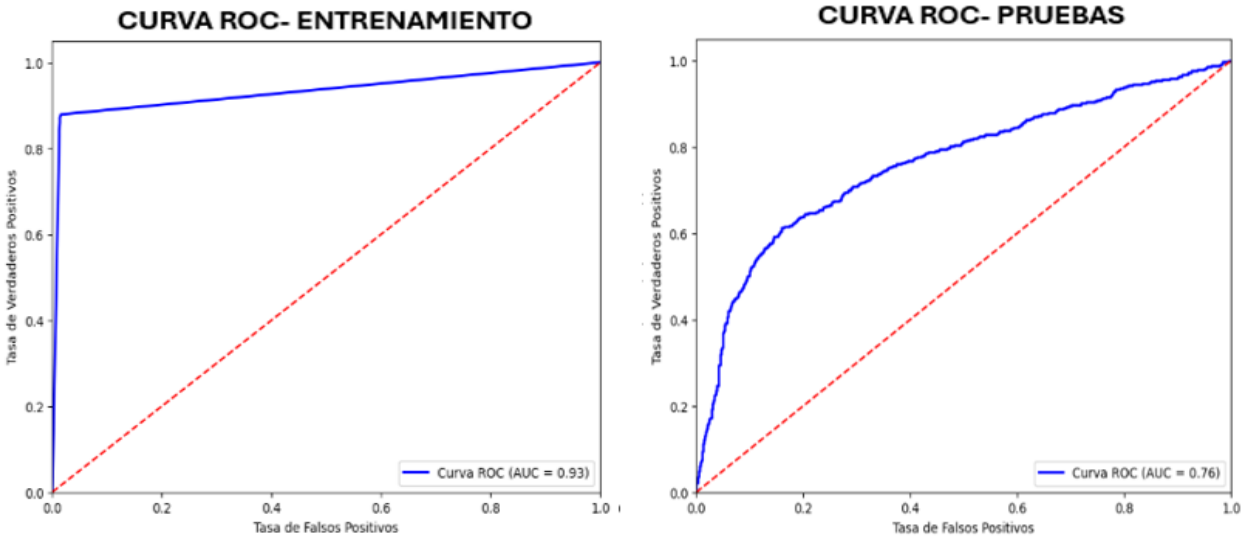
eficazmente entre correos spam y no spam en datos previamente vistos. Sin embargo, en la fase de pruebas reales, el AUC disminuye a 0.76, lo que refleja una caída en la capacidad del modelo para mantener su nivel de discriminación cuando se enfrenta a datos nuevos y más diversos.

Esta diferencia se debe a la forma en que los árboles de decisión identifican patrones en los datos. Durante el entrenamiento, el modelo genera reglas basadas en características específicas que pueden no ser exactamente las mismas en un entorno real, donde los correos electrónicos pueden presentar variaciones en su estructura y contenido. Factores como cambios en el lenguaje, nuevas estrategias de spam y una distribución diferente de los datos pueden influir en su capacidad de clasificación.

A pesar de esta disminución en el AUC, el modelo mantiene un rendimiento aceptable, confirmando su viabilidad como una opción intermedia en la clasificación de correos electrónicos no deseados. Ajustes adicionales en los parámetros del modelo o la combinación con otros enfoques podrían contribuir a mejorar su desempeño en entornos más dinámicos.

Figura 105

Curvas ROC del Modelo Decision Tree en fase de entrenamiento y pruebas

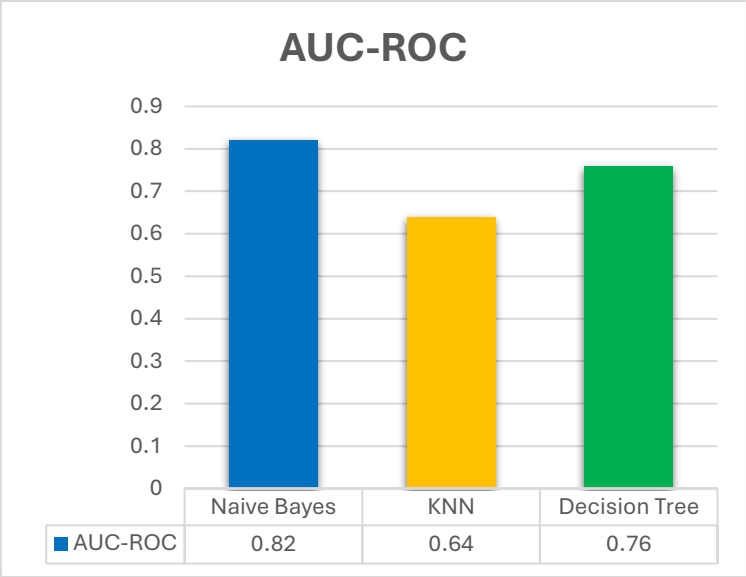


4.1.5.4. Análisis Gráfico

Para facilitar la interpretación de los resultados, se presentaron los gráficos de las métricas AUC-ROC y F1-Score obtenidas en un entorno real. Estos gráficos reflejaron visualmente el desempeño de los algoritmos evaluados, destacando las diferencias en su capacidad para clasificar correos electrónicos como spam o no spam.

Figura 106

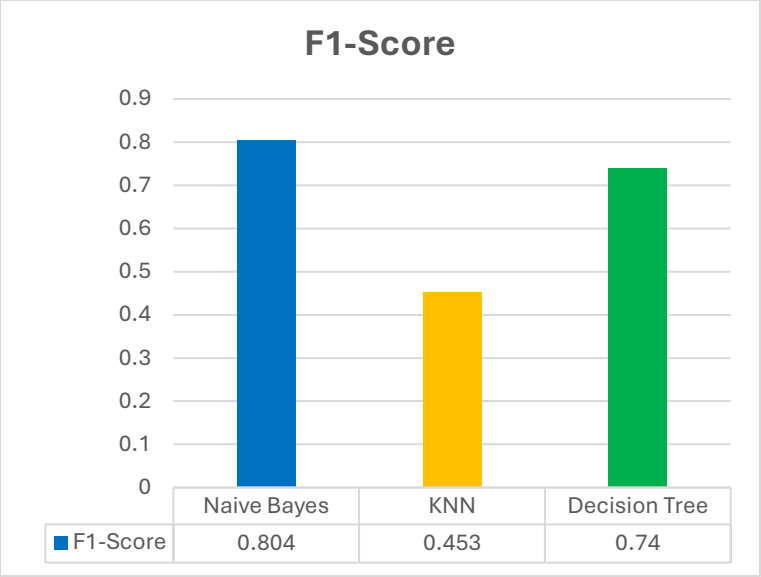
Desempeño de los algoritmos en términos de AUC-ROC en un entorno real



En la Figura 106, el AUC-ROC (Área Bajo la Curva de la Característica Operativa del Receptor) mostró la capacidad de los algoritmos para diferenciar entre las clases de spam y no spam en un entorno real. Un valor más cercano a 1.0 indicó un mejor desempeño. Naive Bayes se destacó con un AUC-ROC de 0.833, seguido por Decision Tree con 0.76, mientras que KNN presentó un desempeño menor con un valor de 0.64. Esto indica que Naive Bayes es el más efectivo en la tarea de clasificación, con una alta capacidad de discriminación entre clases.

Figura 107

Comparación del F1-Score de los algoritmos en un entorno real



La Figura 107 de F1-Score reflejó el balance entre la precisión y el recall de los algoritmos al clasificar correos como spam o no spam en un entorno real. Naive Bayes logró el mayor F1-Score con 0.804, indicando un balance óptimo entre sus métricas. Decision Tree obtuvo un F1-Score de 0.74, mientras que KNN alcanzó un valor más bajo de 0.453 debido a su baja sensibilidad. Estos resultados confirmaron la superioridad de Naive Bayes en escenarios prácticos de clasificación.

4.2. Discusión

Los resultados obtenidos en el presente trabajo demuestran que la aplicación de algoritmos de clasificación para la detección de spam en correos electrónicos permite evaluar su desempeño en distintos escenarios, mejorando la identificación de mensajes no deseados. Estos hallazgos coinciden con los obtenidos por Faris et al. (2019), quienes propusieron un sistema basado en redes neuronales con pesos aleatorios (RWN) y selección de características mediante

algoritmos evolutivos, logrando una mejora en la precisión y reducción del consumo de recursos computacionales.

En el análisis comparativo realizado en el presente estudio, se observó que el algoritmo Naive Bayes presentó un equilibrio entre precisión y recall en entornos de prueba y validación, similar a lo reportado por Faris et al. (2019), quienes encontraron que la selección eficiente de características influye en la reducción de falsos positivos y en la mejora de la clasificación. De igual manera, la implementación del modelo Decision Tree mostró un desempeño estable, destacándose por su robustez en la clasificación de correos electrónicos, lo que coincide con estudios previos como los de Pisani (2023) y Rajesh Kumar et al. (2020) quienes exploraron técnicas de aprendizaje automático para la detección de spam.

Por otro lado, el desempeño del algoritmo K-Nearest Neighbors (KNN) evidenció una alta precisión en la clasificación de correos no spam, pero un recall bajo, lo que sugiere una limitada capacidad de detección de mensajes no deseados. Estos resultados concuerdan con estudios previos como los de Chaudhari et al. (2022) y Sheth et al. (2022), donde se reportó que la efectividad del KNN depende significativamente del valor de 'k' y la distribución de los datos de entrenamiento, afectando su rendimiento en entornos reales.

La integración de técnicas de selección de características, como las propuestas por Chen (2023), ha demostrado ser un enfoque eficaz para mejorar el desempeño de los modelos de clasificación, optimizando el uso de datos relevantes y minimizando el impacto de atributos irrelevantes. En este sentido, los resultados del presente trabajo evidencian que el uso de estrategias de selección de características puede incrementar la eficiencia de los modelos sin comprometer la precisión en la clasificación de correos electrónicos.

Los hallazgos obtenidos en este estudio refuerzan la importancia de la optimización en la selección de características y la elección del algoritmo de clasificación adecuado para la detección de spam. La comparación con estudios previos resalta la validez de los métodos utilizados y su aplicabilidad en escenarios reales, proporcionando un marco de referencia para futuras investigaciones en la mejora de la detección automática de spam mediante aprendizaje automático.

CONCLUSIONES

El presente trabajo ha demostrado la viabilidad de implementar algoritmos de aprendizaje automático en sistemas de detección de spam, evaluando su desempeño mediante la clasificación de correos electrónicos en un entorno controlado y real. Entre los hallazgos más relevantes, se destaca que el algoritmo Naive Bayes fue el modelo más equilibrado, logrando una precisión del 97.17% y un F1-Score de 0.804 en un entorno real. Aunque presentó una disminución en el recall (0.686), mantuvo un balance óptimo entre precisión y sensibilidad, posicionándose como una opción ideal para minimizar tanto falsos positivos como falsos negativos en tareas de clasificación masiva.

Por su parte, KNN, aunque mostró una alta precisión del 100% en la identificación de correos no spam, presentó limitaciones significativas en la detección de correos spam, con un recall de apenas 0.293. Esto resultó en un F1-Score de 0.453, lo que restringe su aplicabilidad en entornos donde la identificación de mensajes spam es prioritaria. En contraste, el algoritmo Decision Tree ofreció un balance razonable entre precisión y recall, con valores de 87.91% y 0.64, respectivamente. Su desempeño intermedio se reflejó en un F1-Score de 0.74, lo que lo convierte en una alternativa viable cuando se busca un equilibrio entre detección de spam y reducción de falsos positivos.

Las pruebas de clasificación realizadas en un entorno real permitieron evaluar el desempeño de los modelos en un conjunto de 2000 correos electrónicos. En este contexto, Naive Bayes clasificó correctamente 1666 correos, mientras que Decision Tree logró 1552 y KNN solo 1293. Además, la evaluación del sistema en condiciones de operación continua permitió procesar un total de 6000 correos electrónicos sin interrupciones, con una tasa de recepción del 100%.

Esto aseguró la fiabilidad del flujo de datos necesario para el procesamiento y clasificación, destacando la robustez del servidor SMTP configurado.

El análisis comparativo evidenció que Naive Bayes y Decision Tree son más adaptables a escenarios complejos, como la presencia de ruido en los datos o características irrelevantes. En cambio, KNN mostró limitaciones en estos casos, afectando su capacidad de generalización. Esto resalta la importancia de seleccionar el algoritmo adecuado según las características del entorno en el que se implementará. Además, la revisión de la literatura realizada en este trabajo permitió establecer una base teórica sólida que guió la implementación y evaluación de los algoritmos seleccionados, consolidando el marco necesario para desarrollar un sistema confiable.

Asimismo, el uso de herramientas de código abierto y servidores robustos garantiza un bajo costo de implementación, haciendo viable este sistema en diversos contextos, como empresas, instituciones educativas o entornos domésticos. El plan de pruebas diseñado fue esencial para validar tanto la funcionalidad básica como el desempeño del sistema en condiciones reales, garantizando que sea confiable y que los resultados sean reproducibles en otros entornos.

Este trabajo contribuye significativamente al desarrollo de sistemas inteligentes de detección de spam, ofreciendo una base sólida para investigaciones futuras que exploren modelos más avanzados, como redes neuronales profundas o enfoques híbridos. En conclusión, los resultados obtenidos confirman que los algoritmos de aprendizaje automático representan una solución eficiente, escalable y económica para la clasificación de correos electrónicos, con posibilidades de adaptación y mejora ante los desafíos futuros de la clasificación en tiempo real.

RECOMENDACIONES

El desarrollo del presente trabajo permitió identificar oportunidades para optimizar y mejorar el sistema propuesto. Una de las principales recomendaciones es explorar el uso de algoritmos más avanzados, como redes neuronales profundas o enfoques híbridos que combinen las fortalezas de Naive Bayes, KNN y Decision Tree. Esto permitiría al sistema enfrentar escenarios más complejos y manejar correos electrónicos con técnicas avanzadas de evasión de spam, aumentando su efectividad en la detección.

Además, se sugiere realizar pruebas adicionales en entornos reales que incluyan un mayor volumen y diversidad de correos electrónicos. Esto permitirá evaluar el desempeño del sistema en condiciones más representativas y realizar ajustes en los parámetros de los algoritmos para maximizar la precisión y la eficiencia. La integración del sistema con plataformas de correo ampliamente utilizadas, como Outlook o Gmail, también es una recomendación importante, ya que proporcionará una validación más sólida de su funcionalidad y compatibilidad en entornos prácticos.

En una implementación en el campo real, sería necesario contar con un experto en traducción para garantizar una adecuada adaptación del dataset a distintos idiomas y contextos. De la misma manera, se recomienda incorporar mecanismos de traducción automatizados que permitan mejorar la detección de spam en cualquier idioma. Esto no solo ampliaría la cobertura del sistema en entornos multilingües, sino que también mejoraría su capacidad para identificar patrones de spam sin depender exclusivamente del idioma en el que esté redactado el correo electrónico.

Dado el carácter dinámico de las técnicas de spam, es esencial incorporar un mecanismo de monitoreo continuo y actualización del sistema. Un módulo de aprendizaje continuo

permitiría que el sistema se adapte automáticamente a nuevos patrones de spam, garantizando su relevancia y efectividad a largo plazo sin la necesidad de intervención manual frecuente. Asimismo, ampliar el conjunto de datos utilizados para las pruebas, incorporando correos electrónicos de diferentes dominios, idiomas y estilos, permitirá fortalecer la robustez y generalización del modelo.

Desde una perspectiva operativa, se recomienda realizar un análisis detallado de los costos a largo plazo asociados con la implementación del sistema, especialmente si se planea utilizar en entornos de gran escala. Esto incluye la optimización de recursos de infraestructura y el mantenimiento periódico para minimizar los costos operativos. En paralelo, es fundamental garantizar la privacidad de los usuarios mediante políticas de manejo seguro de datos, que incluyan cifrado y almacenamiento seguro, cumpliendo con normativas internacionales como el Reglamento General de Protección de Datos.

REFERENCIAS

Albán, F., Urvina, M., & Andrade, R. (2022). Analysis and Design of a Predictive Model for Phishing Detection Based on Url and Email Corpus. *Revista Politecnica*, *50*(3), 27–42. https://doi.org/10.33333/rp.vol50n3.03

Almaguer-Perez, D., & Hernández-Yeja, A. (2021). Buenas prácticas para el despliegue seguro del servicio de correo electrónico. *Revista Científica*, *41*(2), 199–212. https://doi.org/10.14483/23448350.15838

Alpaydin, E. (2014). *Introduction to Machine Learning-The MIT Press*.

Arengas, J., Guzmán, Rafael., & López, M. (2024). Impacto del preprocesamiento en la clasificación automática de textos usando aprendizaje supervisado y reuters 21578. *REVISTA COLOMBIANA DE TECNOLOGIAS DE AVANZADA (RCTA)*, *1*(43), 110–118. https://doi.org/10.24054/RCTA.V1I43.2506

Asamblea Constituyente del Ecuador. (2008). CONSTITUCIÓN DE LA REPÚBLICA DEL ECUADOR. *Registro Oficial*, *449*(20), 25–2021. www.lexis.com.ec

Asamblea Constituyente del Ecuador. (2021). *LEY ORGÁNICA DE PROTECCIÓN DE DATOS PERSONALES*. www.lexis.com.ec

Barrera, G. (2017). *El correo electrónico como herramienta de comunicación interna en una empresa de venta de artículos para el hogar*. http://biblioteca.usac.edu.gt/tesis/16/16_1598.pdf

Barría, F. J. C. (2023). Email Marketing como Influenciador de Compras. *Ciencia Latina Revista Científica Multidisciplinar*, *7*(6), 4280–4290. https://doi.org/10.37811/CL_RCM.V7I6.8993

Bhowmick, A., & Hazarika, S. M. (2018). E-mail spam filtering: A review of techniques and trends. *Lecture Notes in Electrical Engineering*, *443*, 583–590. https://doi.org/10.1007/978-981-10-4765-7_61

Chaudhari, D., Kolambe, D., Patil, R., & Puranik, S. (2022). EMAIL SPAM DETECTION USING MACHINE LEARNING AND PYTHON. En *International Journal of Research Publication and Reviews Journal homepage: www.ijrpr.com* (Vol. 3). www.ijrpr.com

Chen, S. (2023). Multiple Machine Learning Algorithms for Spam Mail Detection. En *Highlights in Science, Engineering and Technology CMLAI* (Vol. 2023).

Chipana, Y. M. M., Osco Escobedo, M. A., Quispe Ichpas, R., Nieto Fernández, G. J., Garcia Quispe, ladys B., & Cerna, D. A. (2023). El Correo electrónico, como medio de intrusión del Phishing y fraude informático. *Revista de Climatología*, *23*, 1138–1148. https://doi.org/10.59427/RCLI/2023/V23CS.1138-1148

Cole, H. L., Hannes, H., & Hapke, M. (2019). *Natural Language Processing in Action*.

Dada, E. G., Bassi, J. S., Chiroma, H., Abdulhamid, S. M., Adetunmbi, A. O., & Ajibuwa, O. E. (2019). Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, *5*(6), e01802. https://doi.org/10.1016/J.HELIYON.2019.E01802

Díez, F. (2021). *¿Qué es acceso al correo mediante IMAP?* https://www.hostinet.com/formacion/correo-electronico/que-es-acceso-correo-mediante-imap/

Dipak. (2018). *POP3 Protocol*. http://solutions24h.com/pop3-protocol/

Faris, H., Al-Zoubi, A. M., Heidari, A. A., Aljarah, I., Mafarja, M., Hassonah, M. A., & Fujita, H. (2019). An intelligent system for spam detection and identification of the most relevant features based on evolutionary Random Weight Networks. *Information Fusion*, *48*, 67–83. https://doi.org/10.1016/j.inffus.2018.08.002

Géron, A. (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow (2019, O'reilly). *Hands-On Machine Learning with R*, 510. https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

Gholamy, A., Kreinovich, V., & Kosheleva, O. (2018). *A Pedagogical Explanation A Pedagogical Explanation Part of the Computer Sciences Commons*. https://scholarworks.utep.edu/cs_techrephttps://scholarworks.utep.edu/cs_techrep/1209

Gómez, Á. (2021). *BIG DATA, UN SISTEMA DE GESTIÓN DE DATOS*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*.

Holt, T. (2019). *Métodos de fraude basados en el correo electrónico*.

Huet, P. (2024, julio 4). *Técnicas clave para el procesamiento de texto en NLP*. https://openwebinars.net/blog/tecnicas-clave-para-procesamiento-texto-nlp/

Linube. (2020). *Cabeceras de correo*. https://linube.com/ayuda/articulo/125/cabeceras-de-correo

Marín, A., Martínez-Merino, L. I., Puerto, J., & Rodríguez-Chía, A. M. (2021). *The soft-margin Support Vector Machine with ordered weighted average*. http://arxiv.org/abs/2107.06713

Mathworks. (2023). *R2023a - Updates to the MATLAB and Simulink product families - MATLAB & Simulink*.

https://www.mathworks.com/products/new_products/release2023a.html?utm_source=chatgpt.com

Mexagon. (2018). *Protocolos de correo electrónico* -. https://apps.mexagon.net/clientes/knowledgebase/218/Protocolos-de-correo-electronico.html

Montero, R. (2020). *Correo IMAP*. www.imap.org

Moreno, G. (2018, julio 27). *Gráfico: ¿Desde qué países se envían más correos basura? | Statista*. https://es.statista.com/grafico/14896/desde-que-paises-se-envian-mas-correos-basura/

Murphy, K. P. . (2012). *Machine learning : a probabilistic perspective*. MIT Press.

Muzammil, M. J., Qazi, S., & Ali, T. (2013). Comparative analysis of classification algorithms performance for statistical based intrusion detection system. *2013 3rd IEEE International Conference on Computer, Control and Communication, IC4 2013*. https://doi.org/10.1109/IC4.2013.6653738

Oracle. (2023). *Java Software | Oracle*. https://www.oracle.com/java/

Pandey, M. (2022, noviembre 24). *8 AI/ML Terms Explained for Beginners*. https://analyticsindiamag.com/ai-trends/8-ai-ml-terms-explained-for-beginners/

Pisani, N. E. (2023). *Técnicas de aprendizaje automático para la detección de spam*. https://repositorio.uai.edu.ar/handle/123456789/1655

Quizhpilema, R. (2023). *Aplicación de la informática forense para identificar la integridad de un correo electrónico.* https://repositorio.upse.edu.ec/handle/46000/10298

Rajesh Kumar, J., Mahalakshmi, G., & Sudarshan, P. (2020). Email Spam Detection using Machine Learning Techniques. *IARJSET*, *8*(6), 189–193. https://doi.org/10.17148/iarjset.2021.8632

República del Ecuador. (2012). *LEY DE COMERCIO ELECTRONICO, FIRMAS Y MENSAJES DE DATOS*. www.lexis.com.ec

República del Ecuador. (2016). *LEY ORGÁNICA DE TELECOMUNICACIONES*.

República del Ecuador. (2021). *CÓDIGO ORGÁNICO INTEGRAL PENAL, COIP*. www.lexis.com.ec

Riabov, V. V. (2017). SMTP (Simple Mail Transfer Protocol). *Handbook of Computer Networks*, *2*, 388–406. https://doi.org/10.1002/9781118256114.CH26

Roshna, S. (2024). *K-Nearest Neighbors Algorithm - Intuitive Tutorials*. https://intuitivetutorial.com/2023/04/07/k-nearest-neighbors-algorithm/

Russell, S., & Norvig, P. (2021). *Artificial Intelligence, Global Edition A Modern Approach*. 1168. https://elibrary.pearson.de/book/99.150005/9781292401171

Scala. (2024). *Introducción | Scala Documentation*. https://docs.scala-lang.org/es/tour/tour-of-scala.html

Sheth, V., Tripathi, U., & Sharma, A. (2022). A Comparative Analysis of Machine Learning Algorithms for Classification Purpose. *Procedia Computer Science*, *215*, 422–431. https://doi.org/10.1016/J.PROCS.2022.12.044

Smarandache, Florentin. (2022). *Collected Papers (On Physics, Artificial Intelligence, Health Issues, Decision Making, Economics, Statistics), Volume XI*. https://papers.ssrn.com/abstract=4316992

Thomas, R., & Shyni, E. (2014). Spam Email Detection using Structural Features. *International Journal of Computer Applications*, *89*(3), 38–41. https://doi.org/10.5120/15485-4265

Uvence Rodríguez, Y. (2020). *Métodos de aprendizaje supervisado para la detección de correos Spam*. https://www.researchgate.net/publication/346551287

Vijayarani, S., & Janani, R. (2016). Text Mining: open Source Tokenization Tools – An Analysis. *Advanced Computational Intelligence: An International Journal (ACII)*, *3*(1), 37–47. https://doi.org/10.5121/ACII.2016.3104

Vujović, Ž. (2021). Classification Model Evaluation Metrics. *International Journal of Advanced Computer Science and Applications*, *12*(6), 599–606. https://doi.org/10.14569/IJACSA.2021.0120670

ANEXOS

ANEXO A: Código de Entrenamiento y Creación de Modelos de Machine Learning para la Detección de SPAM

https://github.com/darwinalmachi/Evaluacion-de-Rendimiento-de-Algoritmos-de-Aprendizaje-en-la-Deteccion-de-Spam/blob/c628fd05f53f6a7ea4d3c6c049e010da134fea0c/ANEXO%20A%3A%20C%C3%B3digo%20de%20Entrenamiento%20y%20Creaci%C3%B3n%20de%20Modelos%20de%20Machine%20Learning%20para%20la%20Detecci%C3%B3n%20de%20SPAM

ANEXO B: Código de Implementación de Modelos de Machine Learning en el Servidor para Filtrado de Correos

https://github.com/darwinalmachi/Evaluacion-de-Rendimiento-de-Algoritmos-de-Aprendizaje-en-la-Deteccion-de-Spam/blob/7f1da87f470925986cc1cd44154882d6e8778e7b/ANEXO%20B%20C%C3%B3digo%20de%20Implementaci%C3%B3nde%20Modelos%20de%20Machine%20Learning%20en%20el%20Servidor%20para%20Filtrado%20de%20Correos.txt

ANEXO C: Análisis de Características y Requisitos para la Comparación de Algoritmos de Aprendizaje Automático

En este anexo, se detallan las características o requisitos considerados para la selección y comparación de algoritmos de aprendizaje automático en la detección de spam. Los criterios clave incluyen simplicidad e intuición, eficiencia para grandes datasets, manejo de datos categóricos y continuos, facilidad de interpretación, baja complejidad computacional y reducción de sobreajuste.

Tabla 17

Desglose de Características por Algoritmo

Características	Descripción
Simplicidad e Intuición	Evaluación de la facilidad de comprensión e implementación del algoritmo. Algoritmos intuitivos y simples de implementar reciben una puntuación de "1".
Eficiencia para grandes datasets	Capacidad del algoritmo para manejar grandes volúmenes de datos de manera eficiente. Algoritmos que manejan eficientemente grandes datasets reciben una puntuación de "1".
Manejo de Datos Categóricos	Aptitud del algoritmo para trabajar con datos categóricos. Algoritmos que manejan datos categóricos reciben una puntuación de "1".
Manejo de Datos Continuos	Capacidad del algoritmo para procesar datos continuos. Algoritmos que manejan datos continuos reciben una puntuación de "1".
Facilidad de Interpretación	Facilidad con la que los resultados del modelo pueden ser interpretados por los usuarios. Algoritmos con resultados fácilmente interpretables reciben una puntuación de "1".
Baja Complejidad Computacional	Nivel de recursos computacionales requeridos por el algoritmo. Algoritmos con baja complejidad computacional reciben una puntuación de "1".

Características	Descripción
Reducción de Sobreajuste	Capacidad del algoritmo para evitar el sobreajuste y generalizar bien a datos no vistos. Algoritmos que reducen el sobreajuste reciben una puntuación de "1".

Nota. Estos criterios permiten una comparación detallada y objetiva de los algoritmos, asegurando que los seleccionados sean los más adecuados para la tarea de detección de spam.

Tabla 18

Características relevantes de los Algoritmos de Aprendizaje Automático

Características	KNN	Naive Bayes	Decision Tree	Logistic Regression	SVM	Random Forest	Redes Neuronales
Simplicidad e Intuición	1	1	1	1	0	0	0
Eficiencia para grandes datasets	0	1	0	0	0	1	1
Manejo de Datos Categóricos	1	1	1	1	0	1	1
Manejo de Datos Continuos	1	0	1	1	1	1	1
Facilidad de Interpretación	1	0	1	0	0	0	0
Baja Complejidad Computacional	0	1	0	0	0	0	0
Reducción de Sobreajuste	0	1	0	0	1	0	0
Cumple “1” No cumple “0”							

Los algoritmos K-Nearest Neighbors (KNN), Naive Bayes y Decision Tree se destacan en estos criterios, lo que los hace especialmente adecuados para la detección de spam (Thomas & Shyni, 2014). KNN es elogiado por su simplicidad, manejo eficaz de datos categóricos y

continuos, aunque puede ser intensivo computacionalmente(Géron, 2019). Por otro lado, Naive Bayes ofrece eficiencia para grandes conjuntos de datos, baja complejidad computacional y buen manejo de datos categóricos, aunque asume independencia entre características (Géron, 2019). Por último, Decision Tree es fácil de interpretar y puede manejar tanto datos categóricos como continuos, aunque puede ser propenso al sobreajuste y menos eficiente con grandes conjuntos de datos(Thomas & Shyni, 2014).

Además de estas consideraciones, se sugiere explorar la combinación de estos algoritmos para mejorar el rendimiento general del modelo, ajustar los parámetros específicos de cada algoritmo para optimizar su rendimiento y evaluar el rendimiento de los algoritmos seleccionados utilizando métricas adecuadas como precisión, tasa de falsos positivos y tasa de falsos negativos(Murphy, 2012).

ANEXO D: Análisis de los Lenguajes de Programación

Este anexo presenta un análisis detallado de las ventajas y desventajas de varios lenguajes de programación relevantes para el desarrollo de sistemas de detección de spam basados en inteligencia artificial. La evaluación incluye lenguajes ampliamente usados como Python, R y Julia, así como otros lenguajes adicionales como MATLAB, Java, Scala, JavaScript (con TensorFlow.js) y C++. Además, se incluye una tabla comparativa que sintetiza las principales características de cada lenguaje en función de criterios clave como facilidad de aprendizaje, soporte de bibliotecas, rendimiento, compatibilidad con IA y ML, y más. Este estudio busca servir como una guía para seleccionar el lenguaje más adecuado en función de las necesidades específicas del proyecto.

Python:

Ventajas:

- **Facilidad de aprendizaje:** Python tiene una sintaxis clara y sencilla, similar al lenguaje natural, lo que facilita su aprendizaje incluso para principiantes(Géron, 2019). Esto reduce el tiempo y los costos de capacitación, permitiendo que los desarrolladores se integren rápidamente al proyecto.
- **Amplio soporte de bibliotecas:** Python cuenta con un ecosistema extenso de bibliotecas y frameworks especializados en inteligencia artificial y machine learning, como TensorFlow, Scikit-learn , Pandas y Numpy. Estas herramientas proporcionan funciones avanzadas para el preprocesamiento de datos, entrenamiento de modelos, evaluación de rendimiento y visualización de resultados.
- **Compatibilidad con IA y ML:** Python es ampliamente utilizado en la comunidad de IA y ML, lo que asegura una fluida integración con tecnologías avanzadas y frameworks de última generación (Géron, 2019). Esto permite aprovechar los últimos avances en el campo para desarrollar un sistema de detección de spam robusto y efectivo.
- **Comunidad activa:** Python cuenta con una gran comunidad activa de desarrolladores y usuarios que ofrecen soporte constante, comparten recursos y colaboran en proyectos (Géron, 2019). Esta comunidad facilita la resolución de problemas, el intercambio de conocimientos y la búsqueda de soluciones a desafíos técnicos.
- Integración con otros sistemas: Python tiene alta compatibilidad con otros lenguajes de programación, bases de datos y sistemas operativos (Alpaydin, 2014). Esto permite una integración flexible y cohesionada del sistema de detección de spam con otros componentes de la infraestructura tecnológica.

- **Documentación y recursos:** Python cuenta con abundante documentación oficial y recursos educativos disponibles en línea, incluyendo tutoriales, libros electrónicos y cursos(Alpaydin, 2014). Esta amplia gama de recursos facilita el aprendizaje del lenguaje, la comprensión de conceptos complejos y la resolución de problemas específicos.

Desventajas:

- **Rendimiento:** El rendimiento de Python puede ser inferior a Julia para tareas computacionalmente intensivas, como el entrenamiento de modelos de aprendizaje profundo con grandes conjuntos de datos(Alpaydin, 2014). Sin embargo, este factor no es crítico para el sistema de detección de spam, ya que las tareas de entrenamiento y clasificación suelen ser manejables dentro de un rango de tiempo razonable.

R:

Ventajas:

- **Análisis estadístico y visualización de datos:** R es reconocido por su excelencia en el análisis estadístico y la visualización de datos (Géron, 2019). Esto lo convierte en una herramienta útil para explorar y comprender los datos de spam, identificar patrones y tendencias, y generar visualizaciones que faciliten la interpretación de los resultados.
- **Amplio soporte de paquetes estadísticos y gráficos:** R cuenta con una amplia biblioteca de paquetes estadísticos y gráficos como ggplot2, dplyr y tidyverse, que proporcionan funciones avanzadas para el análisis de datos, la manipulación de conjuntos de datos y la creación de gráficos informativos.

Desventajas:

- **Versatilidad:** El uso principal de R se limita a análisis estadístico y visualización de datos, reduciendo su versatilidad en comparación con Python, que es adecuado para una variedad de aplicaciones (Alpaydin, 2014).En el contexto del sistema de detección de spam, R no ofrece las mismas capacidades para el desarrollo de modelos de aprendizaje automático, la integración con otros sistemas y la implementación de aplicaciones web.
- **Integración con otros sistemas:** R tiene un menor nivel de soporte para la integración con otros sistemas, bases de datos y frameworks de desarrollo (Alpaydin, 2014). Esto puede generar dificultades al momento de conectar el sistema de detección de spam con otros componentes de la infraestructura tecnológica.

Julia:

Ventajas:

- **Rendimiento:** Julia ofrece un rendimiento muy alto, especialmente en tareas computacionalmente intensivas como el entrenamiento de modelos de aprendizaje profundo (Alpaydin, 2014). Esto podría ser beneficioso para el sistema de detección de spam en caso de manejar grandes conjuntos de datos o mod**elos** complejos.

MATLAB

Ventajas:

- **Facilidad de uso:** MATLAB tiene una curva de aprendizaje relativamente corta, especialmente para usuarios con experiencia en matemáticas y análisis de datos. Su sintaxis intuitiva y herramientas visuales integradas facilitan el trabajo (Mathworks, 2023).

- **Bibliotecas especializadas:** Ofrece bibliotecas robustas para análisis matemático, procesamiento de señales y aplicaciones científicas, lo que puede ser útil en la detección de spam (Mathworks, 2023).
- Soporte de visualización: MATLAB es conocido por sus herramientas avanzadas de visualización, lo que facilita la interpretación de datos y modelos.

Desventajas:

- **Costo:** MATLAB es un software propietario con licencias costosas, lo que puede ser un obstáculo para algunos desarrolladores o equipos pequeños.
- **Rendimiento:** Aunque eficiente en muchas tareas, MATLAB no es tan rápido como lenguajes como Julia o C++ para tareas intensivas en cálculo.
- **Comunidad restringida:** A pesar de ser ampliamente usado en el ámbito académico, su comunidad es menor en comparación con Python o Java.

Java

Ventajas:

- **Portabilidad:** Java es multiplataforma gracias a la máquina virtual Java (JVM), lo que asegura la compatibilidad con diversos sistemas operativos (Oracle, 2023).
- **Rendimiento:** Ofrece un rendimiento aceptable, especialmente con el uso de optimizaciones avanzadas como el compilador Just-In-Time (JIT).
- **Ecosistema maduro:** Posee una gran cantidad de bibliotecas para machine learning e integración con bases de datos.
- **Comunidad activa:** Java cuenta con una comunidad extensa que proporciona soporte constante.

Desventajas:

- **Complejidad:** Su sintaxis es más compleja en comparación con lenguajes como Python, lo que puede dificultar su aprendizaje inicial.
- **Menor especialización en IA:** Aunque es funcional para machine learning, Java carece de bibliotecas tan avanzadas y especializadas como las disponibles en Python.

Scala

Ventajas:

- **Compatibilidad con Java:** Scala es compatible con Java, lo que permite aprovechar bibliotecas y herramientas existentes (Scala, 2024).
- **Programación funcional y orientada a objetos:** Esta combinación permite un código conciso y más expresivo.
- **Escalabilidad:** Diseñado para manejar grandes volúmenes de datos, ideal para aplicaciones distribuidas como sistemas de detección de spam.

Desventajas:

- **Curva de aprendizaje:** Scala es más difícil de aprender debido a su sintaxis avanzada y características funcionales.
- **Comunidad más reducida:** Aunque creciente, su comunidad es más pequeña comparada con lenguajes como Java o Python.

JavaScript (TensorFlow.js)

Ventajas:

- **Ejecución en navegadores:** TensorFlow.js permite ejecutar modelos de machine learning directamente en el navegador, eliminando la necesidad de servidores.
- **Interactividad:** Ideal para aplicaciones web interactivas y en tiempo real.
- **Comunidad:** JavaScript tiene una comunidad global muy activa, con gran cantidad de recursos y soporte.

Desventajas:

- **Rendimiento:** Las tareas complejas de machine learning pueden ser lentas en navegadores comparado con ejecuciones en hardware especializado.
- **Especialización limitada:** TensorFlow.js no tiene el nivel de optimización que ofrecen frameworks como TensorFlow para Python.

C++

Ventajas:

- **Rendimiento:** Es uno de los lenguajes más rápidos debido a su ejecución cercana al hardware, ideal para tareas computacionalmente intensivas.
- **Control:** Ofrece un control completo sobre la gestión de memoria, lo que puede ser útil para optimizar modelos complejos.
- **Bibliotecas de IA:** Frameworks como TensorFlow y PyTorch tienen versiones compatibles con C++ para un rendimiento óptimo.

Desventajas:

- **Complejidad:** Su sintaxis y gestión manual de memoria lo hacen difícil de aprender y usar.

- **Tiempo de desarrollo:** La implementación de soluciones puede tomar más tiempo en comparación con lenguajes como Python.

La elección del lenguaje de programación es crucial para el desarrollo eficiente y efectivo de un sistema de detección de spam utilizando algoritmos de aprendizaje automático. En este estudio, se han evaluado tres lenguajes de programación principales: Python, R, Julia, MATLAB, Java, Scala JavaScript (con TensorFlow.js) y C++. La Tabla 23 resume las características esenciales que se han considerado para la elección del lenguaje de programación.

Tabla 19

Características de los Lenguajes de Programación

Características	Python	R	Julia	MATLAB	Java	Scala	JavaScript (TensorFlow.js)	C++
Facilidad de Aprendizaje	1	1	0	1	0	0	1	0
Amplio Soporte de Bibliotecas	1	1	0	1	1	1	0	1
Compatibilidad con IA y ML	1	1	1	1	1	1	1	1
Rendimiento	1	0	1	0	1	1	0	1
Comunidad Activa	1	1	0	0	1	0	1	1
Integración con Otros Sistemas	1	0	1	0	1	1	1	1
Documentación y Recursos	1	1	0	1	1	1	1	1

Cumple “1” No cumple “0”

ANEXO E: Código Para Envió Masivo de Correos Electrónicos Desde Gmail

https://github.com/darwinalmachi/Evaluacion-de-Rendimiento-de-Algoritmos-de-Aprendizaje-en-la-Deteccion-de-Spam/blob/e1a2465826e9d9c2d33ec425fa6ad9f506825be3/ANEXO%20E%20C%C3%B3digo%20Para%20Envi%C3%B3%20Masivo%20de%20Correos%20Electr%C3%B3nicos%20Desde%20Gmail.txt

ANEXO F: Código Para Generar Mensajes de Spam en Inglés y Español

https://github.com/darwinalmachi/Evaluacion-de-Rendimiento-de-Algoritmos-de-Aprendizaje-en-la-Deteccion-de-Spam/blob/c422cadd9693d3ef90d74ac13a4449873983bd3c/ANEXO%20F%20C%C3%B3digo%20Para%20Generar%20Mensajes%20de%20Spam%20en%20Ingl%C3%A9s%20y%20Espa%C3%B1ol.txt