Facultad de Ingeniería de la Universidad de Buenos Aires

APRENDIZAJE AUTOMÁTICO. TA061

Nombre	**Padrón**	**Mail**
Juan Sebastian Makkos	106229	jmakkos@fi.uba.ar
Sofía Gómez Belis	109358	sgomezb@fi.uba.ar
Matias Agustin Ferrero Cipolla	109886	mferreroc@fi.uba.ar
Gian Luca Spagnolo	108072	gspagnolo@fi.uba.ar
Franco Ricciardo Calderaro	109907	fricciardo@fi.uba.ar
Camila Belen Ayala	107440	cayala@fi.uba.ar

Fecha de Entrega: 26/06/2025

Profesor Titular: MERLINO HERNÁN DANIEL

Índice

Introducción

En la era digital actual, el correo electrónico se ha convertido en una herramienta de comunicación indispensable tanto para fines personales como profesionales. Sin embargo, su omnipresencia también lo ha convertido en un vector común para actividades maliciosas, como el phishing y el spam. Distinguir entre correos electrónicos legítimos y aquellos que buscan engañar o inundar la bandeja de entrada es un desafío constante. Este dataset de correos electrónicos está diseñado para abordar esta problemática, proporcionando un conjunto de datos etiquetados que permiten el desarrollo y la evaluación de modelos de aprendizaje automático capaces de clasificar los correos electrónicos como "Seguros" o "Phishing".

Objetivo

El objetivo de este trabajo consiste no solo en desarrollar un modelo de clasificación automática que permita distinguir entre correos electrónicos legítimos (safe) y correos potencialmente peligrosos (phishing), sino también buscar uno que potencialmente sea el mejor de todos ellos. Para ello, se utilizará un conjunto de datos etiquetado, y se implementará un proceso completo de ciencia de datos que incluye:

- Análisis exploratorio de los datos
- Preprocesamiento y vectorización de texto
- Selección y entrenamiento supervisado de modelos
- Optimización de hiperparámetros
- Evaluación y análisis de los resultados

Para agilizar el proceso y simplificar distintas etapas del modelado (sin perder rigurosidad ni interpretabilidad), se aplicarán herramientas automatizadas de aprendizaje automático como PyCaret. De esta forma no sólo buscamos construir un modelo predictivo efectivo, sino también ilustrar de forma clara las distintas fases del desarrollo de soluciones basadas en machine learning.

Al final, esperamos obtener un modelo con las mejores métricas posibles que nos indiquen su alta capacidad para identificar correos de phishing de manera precisa, minimizando tanto los falsos positivos (marcar un correo safe como phishing) como los falsos negativos (no detectar un correo phishing). Esto permitirá contribuir significativamente a la seguridad digital al reducir la exposición de los usuarios a amenazas de ciberseguridad.

Elección de dataset

Información del Dataset

- Nombre del Archivo: Phishing_Email.csv
- Formato: CSV (Comma Separated Values)
- Contenido: El dataset contiene una colección de correos electrónicos, cada uno clasificado en una de dos categorías: "Safe Email" (Correo Seguro) o "Phishing Email" (Correo de Phishing).

Estructura del Dataset

El dataset está compuesto por las siguientes columnas:

- Email Text: Contiene el cuerpo del texto del correo electrónico. Esta es la característica principal que se utilizará para el análisis y la clasificación.
- Email Type: Esta es la variable objetivo o etiqueta, indicando la clasificación del correo electrónico. Sus posibles valores son:
    - Safe Email: Indica que el correo electrónico es legítimo y no representa una amenaza.
    - Phishing Email: Indica que el correo electrónico es un intento de phishing, diseñado para engañar al destinatario.

Desarrollo

Para comenzar a detallar, lo primero que hicimos fue la carga del conjunto de datos y un análisis preliminar para conocer su estructura, dimensiones y estado general. Estas tareas son fundamentales para entender la naturaleza del problema y preparar los datos para el modelado posterior.

Una vez realizado el primer análisis del dataset y sus datos, lo comenzamos a preprocesar. En esta sección se realiza la limpieza del texto de los correos electrónicos y una básica ingeniería de características. El objetivo es transformar el texto en una representación que pueda ser entendida por los algoritmos de machine learning. Para ello, procedemos a realizar ciertas acciones típicas para PLN (Procesamiento de Lenguaje Natural): en primer lugar, se normaliza y lematiza el texto (la lematización transforma cada palabra a su forma base (ej. "running" → "run"). Esto ayuda a reducir la dimensión del texto sin perder su significado).

Posteriormente, removemos caracteres especiales y codificaciones no estándar. A continuación buscamos convertir las etiquetas (target) a valores numéricos. También convertimos el texto limpio a una matriz numérica (vector) utilizando TF-IDF (Term Frequency - Inverse Document Frequency), que mide la importancia de cada palabra en el conjunto de correos. Por último, se reduce el tamaño del dataset para facilitar el entrenamiento.

Terminada la etapa del preprocesamiento, dividimos el dataset para entrenamiento y validación. Reservamos un 5% del dataset para evaluación posterior (datos no vistos durante el entrenamiento). Esto permitirá validar el desempeño real del modelo. Aquí, es donde comienza la etapa de Entrenamiento. Para ello, comparamos modelos de manera automatizada utilizando PyCaret, que permite comparar múltiples algoritmos de clasificación. El método compare_models() compara automáticamente varios modelos y selecciona el mejor basado en métricas como Accuracy, AUC y F1, entre otras.

Para concluir esta sección, y adelantar un poquito la siguiente, de toda la pool de modelos que fueron entrenados, el mejor fue SVM (Support Vector Machines) con kernel lineal, que corresponde a Stochastic Gradient Descent (SGD Classifier).

Resultados

El clasificador SGD es un algoritmo de clasificación lineal que utiliza el descenso de gradiente estocástico para ajustar un modelo de separación entre clases. Es especialmente eficiente para conjuntos de datos grandes y escasos, como es el caso de texto vectorizado con TF-IDF. En este contexto, el modelo busca encontrar un hiperplano óptimo que separe los correos legítimos de los de phishing en un espacio de características de alta dimensión. El uso de una función de pérdida como la hinge loss permite que el SGD Classifier actúe como un SVM con kernel lineal, lo cual lo hace robusto, rápido y eficaz para tareas de clasificación binaria como esta.

Las métricas más importantes del modelo encontrado gracias a PyCaret (en Entrenamiento) fueron:

- Accuracy: 0.9669
- AUC: 0.9928
- Recall: 0.9811
- Precision : 0.9543
- F1 Score: 0.9674

Sin embargo, tomamos la decisión de utilizar la función tune_model() para encontrar los mejores hiperparámetros del modelo previamente seleccionado. Esto permite mejorar su rendimiento sobre los datos. Una vez satisfechos con el rendimiento del modelo, lo entrenamos con todo el dataset disponible para aprovechar al máximo la información. Sirve

como paso final antes de hacer predicciones en datos nuevos reales o antes de guardar el modelo para producción.

Por último, evaluamos el modelo final sobre el conjunto reservado de datos que no se utilizó en el entrenamiento (data_unseen). Esto simula el comportamiento del modelo ante datos nuevos reales. Se vectoriza el texto con el mismo vectorizador TF-IDF usado en el entrenamiento para mantener consistencia y que el preprocesamiento sea el mismo al utilizado con los datos de train. Sus métricas serán explayadas en la sección de Conclusión.

Conflictos durante el desarrollo

Algunos conflictos que tuvimos a la hora de desarrollar el trabajo fueron tanto las librerías utilizadas, y los recursos físicos necesarios para poder procesar el dataset y entrenar los modelos.

Para las librerías, particularmente las que utilizamos para el procesamiento de lenguaje natural, empezamos usando PyCaret. Pero para PLN lamentablemente esta librería está deprecada, con lo cual cambiamos a NLTK y re (además de las ya conocidas Pandas y sklearn). Ambas son confiables, tienen soporte, y nos sirven para procesar el texto, normalizarlo, limpiarlo, etc. Posteriormente hicimos uso del módulo de clasificación de PyCaret para el entrenamiento.

El otro problema fue el consumo de recursos físicos, sobre todo de memoria RAM a la hora de entrenar los modelos con nuestro dataset. Entrenar modelos es un proceso que requiere demasiada RAM, especialmente si el dataset en el cual se está entrenando es tan extenso como el nuestro. Para eliminar el problema decidimos reducir el dataset aumentando nuestro preprocesamiento sobre el mismo. Al reducir el dataset, el proceso de entrenamiento requirió menos recursos y se pudo llevar a cabo sin problemas.

Conclusión

Para concluir el trabajo práctico y este informe, las métricas obtenidas sobre el conjunto de datos de validación muestran un rendimiento muy sólido de nuestro modelo final:

- Accuracy: El 96% de los correos fueron clasificados correctamente, ya sean phishing o legítimos.
- Precision: Cuando el modelo predice que un correo es phishing, acierta en el 96% de los casos.
- Recall: El modelo detecta correctamente el 96% de los correos phishing reales.

- F1 Score: Un F1 Score de 0.96 indica que el modelo logra un buen equilibrio entre precision y recall, haciendo que sea confiable tanto para detectar amenazas como para no bloquear correos legítimos innecesariamente.

Tal y como esperábamos obtener, hemos llegado a un modelo con una alta capacidad para identificar correos de phishing de manera precisa, minimizando tanto los falsos positivos (marcar un correo safe como phishing) como los falsos negativos (no detectar un correo phishing). Esto permitirá contribuir significativamente a la seguridad digital al reducir la exposición de los usuarios a amenazas de ciberseguridad.