UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

Clasificación de malware metamórfico mediante modelos de aprendizaje automático para facilitar su detección y neutralización

Luis Alejandro Cagigal Camacho

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
INGENIERO EN CIENCIAS DE LA COMPUTACIÓN

Quito, 26 de noviembre de 2023

UNIVERSIDAD SAN FRANCISCO DE QUITO USFQ

Colegio de Ciencias e Ingenierías

HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA

Clasificación de malware metamórfico mediante modelos de aprendizaje automático para facilitar su detección y neutralización

Luis Alejandro Cagigal Camacho

Nombre del profesor, Título académico **Felipe Grijalva Arévalo, Ph.D.**

Quito, 26 de noviembre de 2023

© DERECHOS DE AUTOR

Nombres y apellidos: Luis Alejandro Cagigal Camacho

Código: 00211793

Cédula de identidad: 1718946492

Lugar y fecha: Quito, 26 de noviembre de 2023

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en http://bit.ly/COPETheses.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on http://bit.ly/COPETheses.

RESUMEN

En la era digital actual, la ciberseguridad es fundamental para preservar la integridad de la información en un mundo donde los ciberdelincuentes perfeccionan sus técnicas. El malware metamórfico, con su capacidad para cambiar constantemente, representa un desafío evasivo. Se aborda este problema mediante el aprendizaje automático, destacando las redes neuronales recurrentes y convolucionales unidimensionales. Este trabajo analiza el malware metamórfico mediante un conjunto de llamadas al API de Windows, clasificándolo en ocho tipos. Se emplean técnicas de aprendizaje automático, como preprocesamiento de datos y visualización, para comparar modelos, destacando las redes neuronales recurrentes y convolucionales. Se busca clasificar eficazmente las clases de malware para su detección y neutralización. El resultado más destacable es un modelo LSTM con un 65% de exactitud y valores alrededor del 68% en clasificación multiclase (variantes de F1-score). Se concluye que el modelo LSTM fue estadísticamente el mejor, y se sugiere explorar alternativas para mejorar la exactitud. Este estudio demuestra que el aprendizaje automático es una herramienta efectiva contra el malware avanzado, facilitando su detección y neutralización mediante técnicas de clasificación multiclase.

Palabras clave: ciberseguridad, malware metamórfico, aprendizaje automático, redes neuronales recurrentes, redes neuronales convolucionales, clasificación multiclase.

ABSTRACT

In the current digital era, cybersecurity is essential to preserve the integrity of information in a world where cybercriminals refine their techniques. Metamorphic malware, with its ability to constantly change, poses an elusive challenge. This issue is addressed through machine learning, emphasizing recurrent neural networks and one-dimensional convolutional networks. This study analyzes metamorphic malware using a dataset of Windows API calls, classifying it into eight types. Machine learning techniques, such as data preprocessing and visualization, are employed to compare models, highlighting recurrent neural networks and one-dimensional convolutional networks. The goal is to effectively classify malware classes for detection and neutralization. The most notable result is an LSTM model with 65% accuracy and values around 68% in multiclass classification (F1-score variants). It is concluded that the LSTM model was statistically the best, and suggestions are made to explore alternatives for accuracy improvement. This research demonstrates that machine learning is an effective tool against advanced malware, facilitating its detection and neutralization through multiclass classification techniques.

Key words: cybersecurity, metamorphic malware, machine learning, recurrent neural networks, one-dimensional convolutional networks, multiclass classification.

TABLA DE CONTENIDO

ÍNDICE DE TABLAS 8
ÍNDICE DE FIGURAS 9
INTRODUCCIÓN 10
DESARROLLO DEL TEMA 12
Estado del arte y revisión de conceptos importantes 12
Explicación del dataset y visualización 16
Preprocesamiento e implementación de modelos de aprendizaje automático 19
Evaluación de modelos y comparación de resultados 29
Análisis estadístico y pruebas de hipótesis 34
Figuras y métricas adicionales del modelo de mejor rendimiento 35
CONCLUSIONES 39
REFERENCIAS BIBLIOGRÁFICAS 41

ÍNDICE DE TABLAS

Tabla 1. Resumen de resultados obtenidos. …… 34

Tabla 2. Reporte de clasificación para el modelo LSTM. …….. 38

ÍNDICE DE FIGURAS

Figura 1. Promedio de llamadas más frecuentes para cada clase. ……. 17

Figura 2. Longitud de secuencias promedio, mínima longitud y máxima longitud para cada clase. …….. 18

Figura 3. Visualización del dataset completo, utilizando t-SNE. …….. 19

Figura 4. Esquema de preprocesamiento para implementación de SVM. ……. 21

Figura 5. Características seleccionadas por SelectKBest, basado en f_score. ……... 21

Figura 6. Visualización del dataset con t-SNE luego de la selección de características. ... 22

Figura 7. Resumen de la implementación del modelo SVM. ……. 23

Figura 8. Esquema de preprocesamiento para los modelos de redes neuronales. …….. 24

Figura 9. Arquitectura del modelo LSTM. ……. 25

Figura 10. Arquitectura del modelo 1D-CNN. ……... 26

Figura 11. Arquitectura del modelo 1D-CNN-LSTM. ……... 28

Figura 12. Arquitectura del modelo GRU. ……. 29

Figura 13. Metodología de implementación de transfer learning. ……. 30

Figura 14. Comparación entre la precisión de los conjuntos de entrenamiento y validación para optimización de hiperparámetros. ……. 32

Figura 15. Resumen de resultados para entrenamiento y prueba. …….. 35

Figura 16. Resultados de Macro F1, Micro F1 y Weighted F1. …… 35

Figura 17. Gráficos de la función de costo y exactitud del modelo por cada época de entrenamiento. ……... 37

Figura 18. Matriz de confusión del modelo LSTM. …….. 38

INTRODUCCIÓN

En la era digital actual, la ciberseguridad se ha convertido en un componente esencial para la preservación de la integridad y la confidencialidad de la información en todo el mundo. Sin embargo, a medida que la tecnología avanza, los ciberdelincuentes también perfeccionan sus técnicas para eludir las defensas cibernéticas y llevar a cabo ataques cada vez más sofisticados. Entre estas amenazas, el malware metamórfico se destaca como un desafío especialmente complejo y evasivo.

El malware metamórfico es una categoría de software malicioso que se caracteriza por su capacidad para cambiar constantemente su forma y comportamiento, lo que dificulta su detección y análisis por parte de las soluciones de seguridad tradicionales, basadas en comportamientos y firmas digitales. Estos programas maliciosos utilizan técnicas de ofuscación avanzada y mutación para modificar su código y comportamiento con cada infección, lo que hace que sean extremadamente resistentes a las firmas de malware estáticas y a las técnicas de detección convencionales.

La lucha contra el malware metamórfico ha llevado a una creciente necesidad de enfoques más avanzados y automatizados para su identificación y clasificación. En este contexto, el aprendizaje automático (machine learning) ha emergido como una herramienta poderosa para analizar y categorizar estas amenazas evasivas. Mediante el entrenamiento de modelos de machine learning con conjuntos de datos de malware metamórfico, se ha logrado un progreso significativo en la detección y clasificación precisa de estas variantes maliciosas.

Durante el desarrollo de este trabajo, se analizarán los comportamientos del malware metamórfico utilizando un dataset que contiene un conjunto de llamadas secuenciales al API de Windows, y se lo clasificará en ocho tipos diferentes: “spyware”, “downloader”, troyanos, gusanos, “adware”,

“dropper”, virus y “backdoor”. Se utilizarán técnicas de aprendizaje automático como preprocesamiento de datos, visualización, implementación de modelos y análisis estadísticos para comparar dichos modelos; entre los cuales se destacan las redes neuronales recurrentes (RNNs) y redes neuronales convolucionales unidimensionales (1D-CNN), además de un modelo SVM (Support Vector Machines) como base de estudio.

A lo largo de este trabajo se buscará clasificar con efectividad las clases de malware para facilitar su detección general y neutralización, utilizando un acercamiento prometedor como lo es el aprendizaje automático, comprender de mejor manera el comportamiento general del malware metamórfico y contribuir al desarrollo continuo en el área de la ciberseguridad.

DESARROLLO DEL TEMA

Estado del arte y revisión de conceptos importantes

Antes de comenzar con cualquier clase de desarrollo, se requiere una investigación previa del estado del arte para obtener la mayor cantidad de conceptos importantes a tener en cuenta con el objetivo de enfocar los esfuerzos en relación a ellos. Debido a que este trabajo está centrado alrededor del malware metamórfico, es sumamente importante explorar este concepto a pesar de que ya fue mencionado brevemente en la sección introductoria. Además a este concepto, se explorarán otros relacionados al ámbito del aprendizaje automático, ya sean modelos, herramientas, interpretaciones, etc.

Malware metamórifco.

Antes de definir concretamente al malware metamórifco, es importante comprender el término *malware*. Este se define como cualquier tipo de software que es desarrollado con intenciones maliciosas, también conocido como software malicioso, tiene como objetivo robar, ingresar, atacar o destruir a determinados objetivos según sus características. A partir de esto, se puede definir al malware metamórifco como un tipo de malware avanzado; es decir que comparte los mismos objetivos que malware tradicional, pero es capaz de alterar su comportamiento y su estructura para evitar métodos de detección tradicionales, basados en precisamente comportamientos (heurística) y firmas digitales. Como expresa Brezinski (Brezinksi, 2023), esta clase de malware funciona a través de un motor metamórfico que se encarga de la ofuscación y reconstrucción del mismo con el objetivo de que cada generación tenga un diferente comportamiento al anterior, evitando detección por firmas digitales. Existen otros tipos de malware avanzado como el malware polimórfico o el malware oligomórfico; para los cuales las técnicas tradicionales para su detección

y neutralización funcionan efectivamente. La gran diferencia de estas clases de malware con el malware metamórfico reside en el motor metamórfico como tal. Por ejemplo, para malware polimórfico y oligomórfico, el código maligno estaría encriptado y presente fuera de un código benigno, aislado, lo que requeriría una secuencia de descifrado aparte para la ejecución de la sección maligna del código. En cambio, para malware metamórfico, el código maligno está presente en el código benigno como tal, de igual forma encriptado, pero ofuscado de tal manera que una sola secuencia de descifrado es necesaria para la ejecución del código general. Para que todo esto funcione, el motor metamórfico es el encargado de la ofuscación y cifrado del malware, incluyendo su ensamble, desensamble, permutaciones, expansiones y reducciones. El comportamiento de esta clase de malware está explicado con mayor detalle en la cita de Brezinski (Brezinski, 2023).

Aprendizaje automático.

El aprendizaje automático, también conocido como *machine learning*, es una de las ramas de la computación que más ha crecido en los últimos años, a pesar de haber sido propuesta hace ya varias décadas. El poder computacional que existe en la actualidad ha permitido a los algoritmos desarrollarse, probarse y evaluarse de manera efectiva en los años recientes, por lo que muchos problemas computacionales pueden ser abordados con técnicas de este estilo. Existen varios "sabores" del machine learning como tal, que abarcan dentro de si estrategias, modelos, algoritmos y fórmulas, pero este trabajo estará centrado concretamente en aprendizaje supervisado por la naturaleza del dataset; el cual se caracteriza por poseer una columna que especifica a que clase pertenece cada secuencia de llamadas de API.

Existen dos tipos de clasificación, binaria y multiclase. El dataset que se va a utilizar posee un total de ocho clases diferentes, por lo que se abordará el problema como clasificación multiclase. Dentro de los algoritmos de aprendizaje supervisado, existen los modelos que se utilizarán para resolver nuestro problema de clasificación. Estos son SVMs (Support Vector Machines), RNNs (Recurrent Neural Networks) y 1D-CNNs (One-dimensional Convolutional Neural Networks). Según Akinsola (Akinsola, 2017), las SVMs se basan en la noción de una especie de margen que separa dos clases en un hiperplano, y así diferenciar los conjuntos de datos.

Las RNNs son una de las múltiples variantes de redes neuronales que existen, pero más concretamente, estaríamos hablando de un modelo conocido como LSTM (Long Short Term Memory), Esta clase de modelo es bastante popular para interpretar y resolver problemas basados en secuencialidad, debido a sus características, mencionado por Staudemeyer (Staudemeyer, 2019). Una de sus múltiples aplicaciones es su extensa contribución al campo de NLP (Natural Language Processing). Una variante a este modelo sería una red neuronal convolucional de una dimensión (1-D CNN), que contribuye igualmente a la resolución de problemas basados en secuencialidad. El estudio de estos modelos se explica más en la siguiente sección.

Estado del arte y trabajos relacionados

Dentro del área del machine learning y la ciberseguridad hay varias publicaciones que han sido útiles para determinar conceptos, técnicas, modelos, y demás para el desarrollo del proyecto. Por ejemplo, los artículos publicados que se están utilizando principalmente como base de estudio proponen modelos y arquitecturas prometedores en el análisis de secuencias de malware. Por ejemplo, Catak et al (Catak, 2020) propone un modelo LSTM sobre métodos tradicionales de estudio como SVM y RFC, incluso logrando resultados de alrededor del 45% de exactitud para el contexto de clasificación multiclase para el mismo dataset que se estudiará en este proyecto. Cabe

mencionar que presenta resultados de clasificación binaria de alrededor del 90% de exactitud, pero analizando los resultados del F1-score, se encontró que existe un problema relacionado con lo desbalanceadas que estaban las muestras al momento del entrenamiento; que, si bien logran el objetivo propuesto en su trabajo, no sigue el mismo objetivo de este proyecto. Por ende, se tomará en cuenta los resultados de la clasificación multiclase como referencia. Otro ejemplo en cuanto a análisis de secuencias de llamadas por malware, es el trabajo de Le, Boydell, Mac Namee y Scanlon (Le, 2018), que proponen arquitecturas de 1D-CNN (CNN unidimensional) y LSTMs unidireccionales y bidireccionales. A pesar de trabajar con un dataset diferente, logra resultados competentes de alrededor del 95% de exactitud. Cabe mencionar que las secuencias para este dataset ya fueron filtradas y preprocesadas. Otra publicación que propone CNN-LSTM como arquitectura para análisis de secuencias es el de Akhtar y Feng (Akhtar, 2022), logrando valores de exactitud de alrededor del 90%, igualmente para un dataset diferente enfocado en la clasificación de secuencias benignas y malignas (binario). Otra arquitectura interesante propuesta para el análisis de malware en Android es la GRU. En el trabajo de Lu, Du, Ouyang, Chen y Wang (Lu, 2020) proponen arquitecturas de GRU de aprendizaje profundo. Debido a que esas son las arquitecturas más comunes, se analizarán las mismas en el desarrollo de este proyecto.

Relevancia y contribución

A nadie le gustaría que un malware entre en su dispositivo y realice tareas que puedan comprometer la integridad de datos sensibles, archivos importantes y componentes de hardware comprometidos. Por ello, la detección de malware y el mercado de los antivirus es muy popular en la actualidad. La manera en la que muchos antivirus trabajan, es detectar comportamiento que pueda ser sospechoso mediante constantes escaneos al sistema operativo. En caso de detectar algo, se pone en ejecución las contramedidas para neutralizar el comportamiento. Dentro de este

proceso, se define que clase de malware es, debido a que cada clase de malware realiza acciones específicas y necesita de acciones específicas para ser neutralizado. Este proceso de identificación puede tomar tiempo considerable, que puede ser vital para combatir malware avanzado. El uso de inteligencia artificial para facilitar esta detección puede mejorar significativamente esta clase de procesos, haciendo a las antivirus herramientas mucho más capaces. Este trabajo busca contribuir a la optimización de estos procesos, fomentando la investigación y la experimentación para obtener resultados que se alineen con los objetivos generales en cuanto a la detección, clasificación y neutralización de malware avanzado.

Explicación del dataset y visualización

El dataset que se utilizará durante este trabajo está referenciado en [4, 5]. Contiene una única columna que posee las secuencias de llamadas al API de Windows, obtenidas a través del API de *VirusTotal* generado dentro de *Cuckoo Sandbox*. Los autores del dataset lograron de construir estas secuencias y clasificarlas en ocho tipos de malware diferentes:

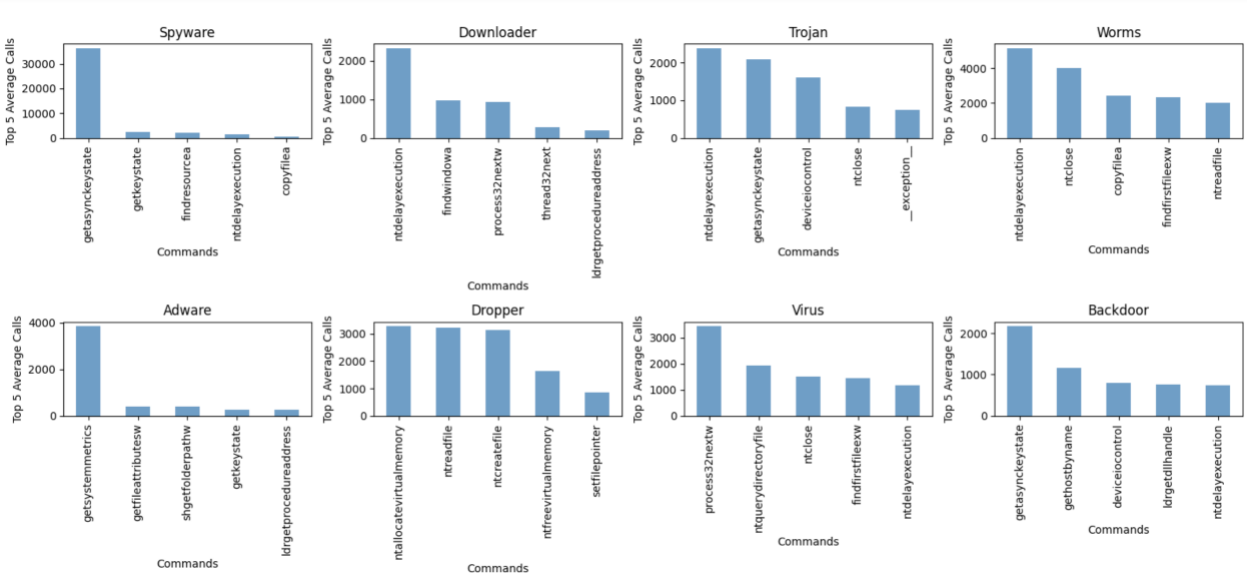
- *Spyware:* transmite información acerca de ciertas acciones de un objetivo de manera oculta hacia el atacante.
- *Downloader:* descarga elementos hacia el objetivo.
- *Trojan:* software que permanece inactivo hasta que cierta acción es ejecutada, oculta su verdadero propósito.
- *Worms:* se replican continuamente en una multitud de objetivos, común con ataques DDoS.
- *Adware:* su único propósito es inundar al objetivo con anuncios.
- *Dropper:* traslada otros tipos de malware hacia objetivos para que sean ejecutados.

- *Virus:* se reproducen entre varios objetivos, y puede replicarse, se adjuntan a ejecutables.
- *Backdoor:* proporciona una entrada al sistema de seguridad del objetivo sin ser detectado para acceder a información.

Cada entrada del dataset está conformada por una secuencia de comandos, o llamadas al API, que fueron recuperadas de un *Cuckoo Sandbox Environment*, y se ven de esta forma: "ldrgetdllhandle ldrgetprocedureaddress getsystemtimeasfiletime...". Estas secuencias tienen longitudes variables y existen un total de 7017 entradas que abarcan las ocho clases. Lo último importante a mencionar del dataset como tal es que está desbalanceado, es decir que existen más entradas para ciertas clases. Por ejemplo, la clase *Adware*, solo tiene 379 entradas, mientras que algunas de las otras clases poseen hasta 1001 entradas.

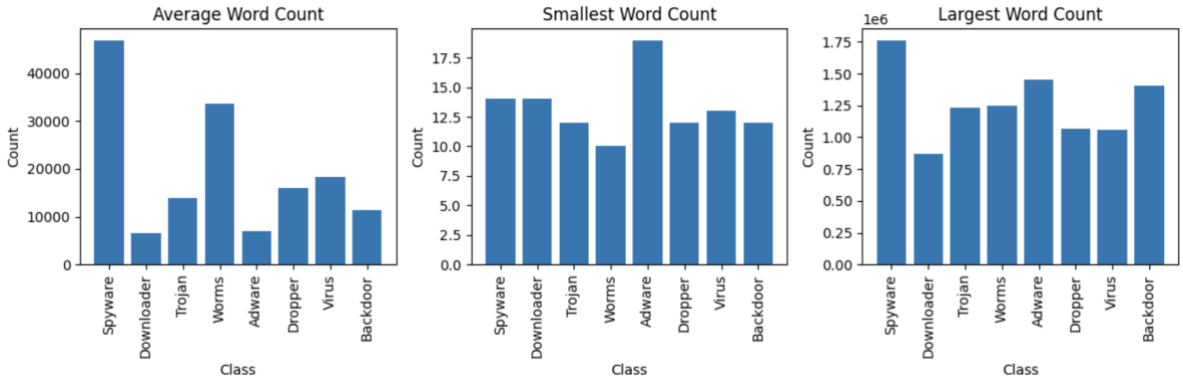
La visualización del dataset puede interpretarse de diferentes maneras, por lo que a continuación, se describirán algunas figuras que representan al dataset de manera más gráfica, para que sea más fácil de comprender desde afuera su estructura.

Figura 1. Promedio de llamadas más frecuentes para cada clase.



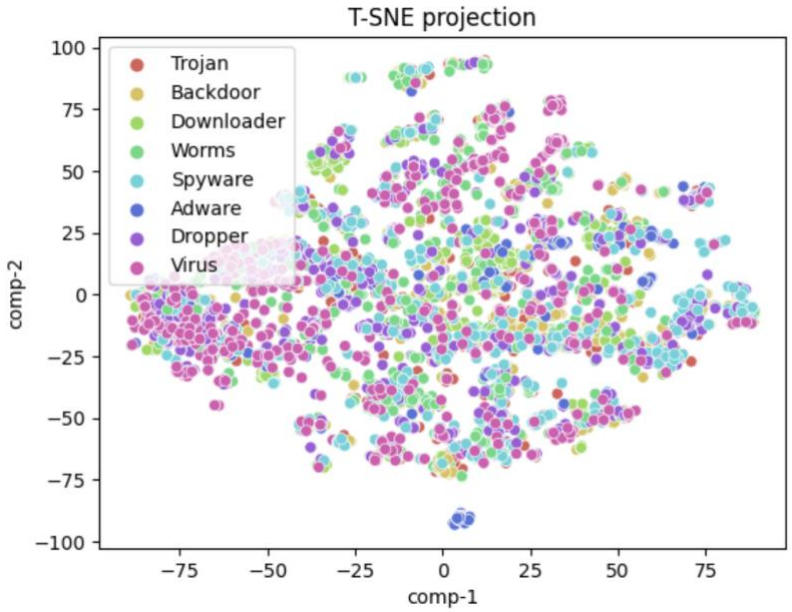
La figura 1 muestra los promedios de las llamadas más frecuentes para cada clase, según cada entrada. Por ejemplo, se puede ver que hay llamadas que destacan significativamente sobre otras en las clases de *Spyware* y *Adware*, mientras que en las demás clases está un poco más nivelado. Con esto, se puede determinar alguna clase a simple vista por las llamadas que posee, pero sigue siendo información que podría no determinar nada concreto.

Figura 2. Longitud de secuencias promedio, mínima longitud y máxima longitud para cada clase.



La figura 2 muestra, en cambio, la longitud promedio de secuencias de llamadas o comandos, para cada clase. También muestra la longitud más pequeña, y más grande para esas mismas clases. A simple vista, se puede determinar que las secuencias más extensas en promedio son las de la clase *Spyware* y *Worms*, mientras que las más pequeñas están dentro de la clase *Downloader* y dentro de la clase de *Adware*. Nuevamente, es información que puede ser útil para visualizar e interpretar de mejor manera el dataset, sin embargo, no es concluyente, por lo que utilizar técnicas de machine learning puede significar la superación de estas barreras.

Figura 3. Visualización del dataset completo, utilizando t-SNE.



Para la figura 3, se utilizó una técnica común para visualización de datos de alta dimensionalidad, conocida como t-SNE (t-Distributed Stochastic Neighbor Embedding). Es bastante popular para encontrar patrones y posibles relaciones no lineales entre datasets de alta dimensionalidad. En este caso, no ofrece demasiada información ya que cada punto (que representa una entrada del dataset), está demasiado cerca a otros sin definir ciertos límites que podrían existir. Más adelante, se utilizará esta misma técnica para visualizar el dataset luego del proceso de selección de características para el modelo de SVM.

Analizando el dataset en temas de frecuencias de llamadas individuales, se han identificado patrones para algunas de las clases. En general, existen un total de 278 llamadas únicas, por lo que para los métodos tradicionales como SVM, RFC y modelos similares, se necesitaría realizar un vector de frecuencias (TF-IDF) para obtener que llamadas individuales se repiten continuamente en las secuencias.

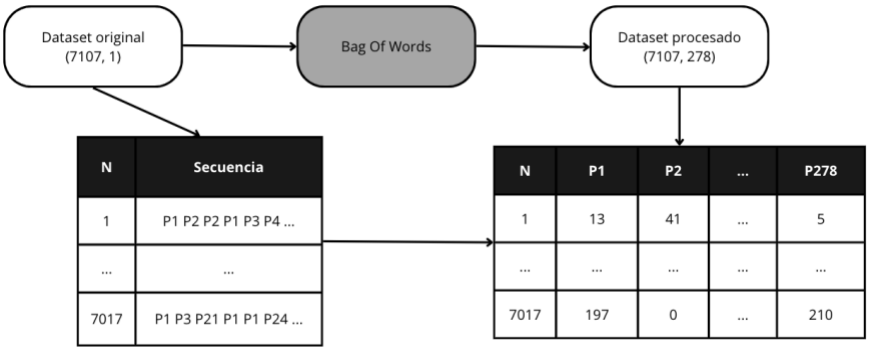
Preprocesamiento e implementación de modelos de aprendizaje automático

Modelo base: SVM (Support Vector Machines).

Como modelo base, se ha escogido implementar un modelo de SVM. Akinsola (2017) expresa algunas de las características que se atribuyen a este modelo, las cuales ya fueron mencionadas previamente en la sección de definición de conceptos. Este modelo existe dentro de la librería de "sklearn" para Python. Antes de ingresar los datos de entrenamiento al modelo, se necesita un preprocesamiento de los datos. Una vez realizado el preprocesamiento, se avanza hacia un proceso de selección de características para reducir la dimensionalidad de los datos. El siguiente paso es la optimización de hiperparámetros del modelo y finalmente se entrena el modelo.

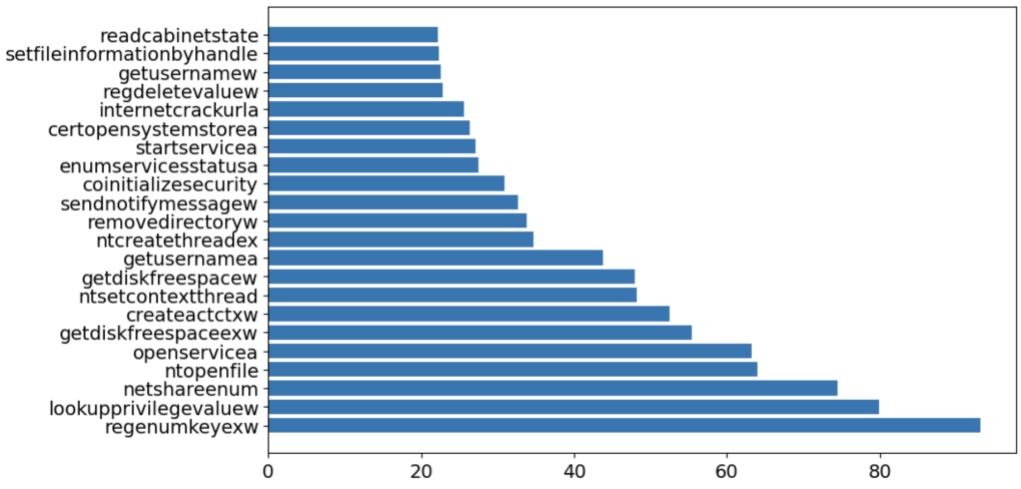
El preprocesamiento de los datos es fundamental al momento de realizar la implementación de un modelo de machine learning. Esto se hace con el fin de ajustar los datos para que el modelo pueda leerlos sin inconvenientes, y también se hace con el objetivo de simplificar el aprendizaje del modelo para obtener mejores resultados. Para el preprocesamiento de este modelo, se utilizó una técnica conocida como Bag of Words, formando lo que se conoce como un vector de frecuencias (TF-IDF), que cuenta las iteraciones de llamadas únicas en cada una de las secuencias. Al ser un dataset que contiene secuencias de texto, hay que transformar los datos de forma numérica. El acercamiento Bag of Words consiste en contar las iteraciones de cada palabra (en este caso sería cada llamada al API), y montar una matriz con cada característica conformada por las 278 llamadas únicas, y contando sus iteraciones en cada entrada; transformando el dataset de tamaño original (7017, 1) a (7017, 278). Se puede evidenciar mejor en la figura 4:

Figura 4. Esquema de preprocesamiento para implementación de SVM.



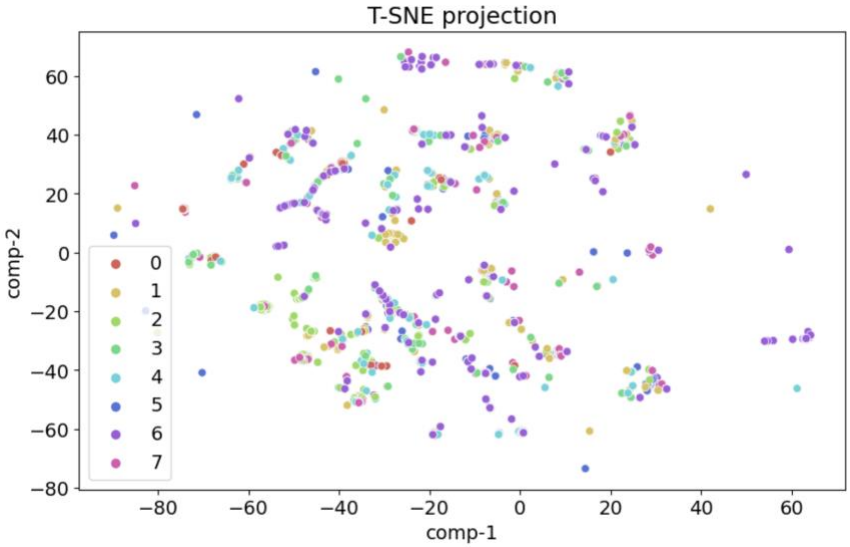
Una vez conseguida la adaptación del dataset, el siguiente paso es seleccionar las características más importantes. Se utilizó un método de filtrado a través de 'f_score'. Esto significa que las columnas (procesos) que maximicen este valor serán las seleccionadas. Este proceso se hace para reducir complejidad y tiempos de ejecución al momento de entrenar el modelo. Este proceso verifica qué columnas influyen más en los resultados, y selecciona un número definido de las mismas. Para este caso, el número de características definido está entre las 20 y 30 características. En la figura 5 se pueden ver las características escogidas luego del filtro:

Figura 5. Características seleccionadas por SelectKBest, basado en f_score.



También se puede hacer nuevamente el proceso de visualización con t-SNE, únicamente con las características seleccionadas para ver si pueden determinarse patrones más claros ahora (comparar con la figura 3).

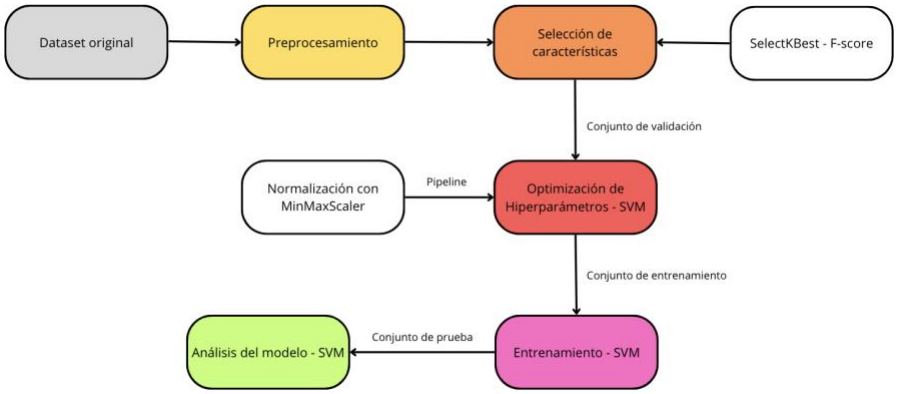
Figura 6. Visualización del dataset con t-SNE luego de la selección de características.



El siguiente paso en la implementación del modelo es la optimización de hiperparámetros. Cada modelo puede representarse como un conjunto de expresiones matemáticas, por lo que tienen algunos valores que deben ser definidos explícitamente. Para el caso de SVM, los hiperparámetros se constituyen por: kernel, C y gamma. Para el caso del kernel, se está utilizando uno gaussiano (‘rbf’); el kernel también puede ser lineal o cuadrático según los bordes que se quieran definir. Utilizando GridSearchCV (Cross Validation) podemos entrenar el modelo y probar diferentes conjuntos de hiperparámetros utilizando un conjunto de validación. Para optimizar estos hiperparámetros hay que tomar en cuenta factores como overfitting, underfitting, así como precisión y otras métricas que determinarán la efectividad del modelo. Para la implementación de esta sección, se utilizó un Pipeline, que simplifica sustancialmente el código. Dentro de este elemento, se añadió un MinMaxScaler, que normalizará los datos para que los valores numéricos estén situados entre 0 y 1, con el fin de simplificar el aprendizaje del modelo, ya que es más fácil para el mismo aprender con datos normalizados.

Una vez obtenidos los hiperparámetros que mejores resultados de validación cruzada retorno, los utilizamos para instanciar el modelo y entrenarlo con su respectivo conjunto de entrenamiento, y analizaremos sus resultados de precisión en el conjunto de prueba.

Figura 7. Resumen de la implementación del modelo SVM.



LSTM (Long Short-Term Memory) y LSTM Bidireccional

Para el modelo de LSTM y los próximos modelos de redes neuronales, se utilizó la librería de Python conocida como 'Keras', así como 'TensorFlow', debido a la amplia utilidad que brindan al momento de crear, editar y optimizar arquitecturas de redes neuronales. Todos los modelos siguen el mismo proceso de implementación, expresado como preprocesamiento y adaptación del dataset, separación del dataset en conjuntos de entrenamiento, prueba y validación, implementación del modelo, optimización de hiperparámetros con el conjunto de validación, entrenamiento del modelo con el conjunto de entrenamiento y evaluación de resultados con el conjunto de prueba.

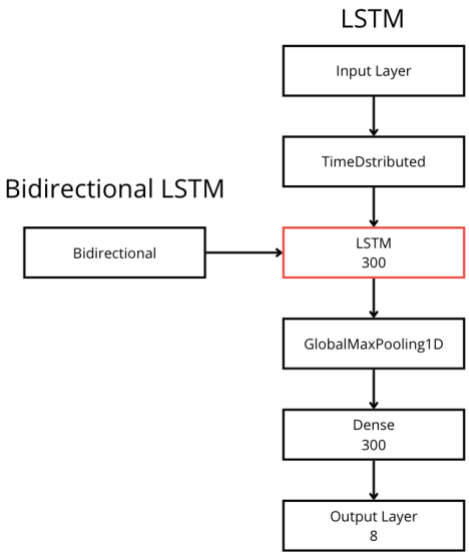
Para el preprocesamiento, se siguió con un procedimiento estándar para transformar secuencias para esta clase de modelos. Se siguió la guía constante de artículos académicos que seguían este mismo procedimiento y foros como "StackOverflow" para responder a dudas específicas. Lo primero que se hizo fue tokenizar las secuencias, de tal forma que se transforma el texto en un

objeto de secuencia formal. Debido a que las secuencias son de longitud variable, luego se procedió a hacer lo que se conoce como ‘padding’, para reducir el tamaño máximo de las secuencias (en este caso 2000) e igualar todas las secuencias a un mismo tamaño; se puede observar el esquema del proceso en la figura 8.

Figura 8. Esquema de preprocesamiento para los modelos de redes neuronales.

Una vez cumplido el preprocesamiento del dataset, se realiza la implementación del modelo junto con la optimización de hiperparámetros. Para el caso de LSTM, se optimizarán el número de capas ocultas de la red y el número de unidades LSTM. Cumplido este proceso, en la figura 9 se muestra el esquema que otorgó los mejores resultados posterior a la optimización:

Figura 9. Arquitectura del modelo LSTM.



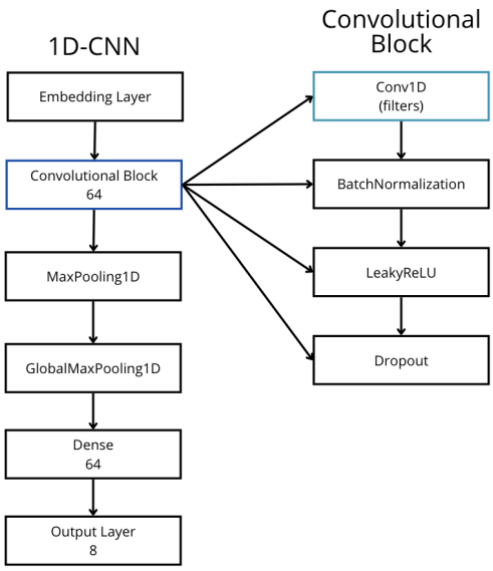
La figura 9 muestra la arquitectura del modelo de LSTM, en donde se puede observar que se optó por una capa ‘TimeDistributed’ para establecer estampas de tiempo en las secuencias. Luego de la optimización, la configuración que mostró mejores resultados fue una sola capa de LSTM, con 300 unidades, seguida por una capa de ‘GlobalMaxPooling1D’ con el objetivo de extraer los pesos más significativos, reduciendo el tiempo de entrenamiento y maximizando los resultados. Luego, una capa densa de 300 neuronas para ajustar los resultados obtenidos del ‘pooling’ seguida por la capa de salida. En la capa densa previa a la salida se optó por una función de activación ‘relu’; justificada por la literatura en los modelos investigados. En la capa de salida, al ser un problema multiclase, se optó por la función de activación ‘softmax’, que retorna un vector de probabilidades (el cual su suma es igual a 1) ajustada a las 8 clases. Si bien hay otras formas de conformar una arquitectura de LSTM, se encontró que esta configuración ofrecía los mejores resultados.

Para el modelo bidireccional; la gran diferencia se encuentra en la manera en que se toman comandos como contexto. Es decir que los modelos bidireccionales, en teoría, entienden mucho mejor los contextos de las oraciones dentro del NLP. La idea aquí es que una LSTM unidireccional solo interpreta los datos en una sola dirección; de inicio hasta el final. Los modelos bidireccionales Interpretan los datos desde el inicio hasta el final, y de regreso; desde el final hasta el inicio. Este acercamiento ha logrado buenos resultados en datasets conocidos (Graves, 2013) y pueden ser una alternativa prometedora para resolver el problema de este proyecto. El esquema del modelo bidireccional puede observarse en la figura 9. Sigue exactamente el mismo esquema que una LSTM unidireccional, la única diferencia es que la capa LSTM es envuelta por una capa bidireccional.

1D-CNN (One-dimensional Convolutional Neural Network) y 1D-CNN con LSTM

Nuevamente, se repite el proceso habitual para la implementación de los modelos. Para el preprocesamiento, se siguió el mismo método mencionado para la LSTM, expresado en la figura 8. Se separa en los conjuntos de entrenamiento, validación y prueba. Se procede con la implementación del modelo como tal, mostrando en la figura 10 su arquitectura:

Figura 10. Arquitectura del modelo 1D-CNN.

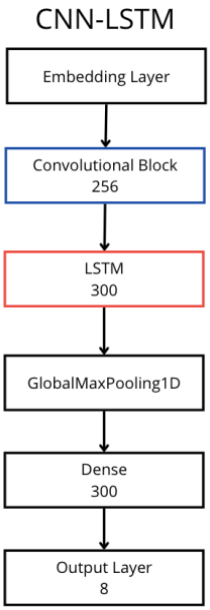


Como se observa en la figura 10, la capa de entrada es un Embedding, el cual transforma las secuencias preprocesadas en vectores codificados (mediante “one-hot encoding”). Esto permite que los datos ingresen sin problema al bloque convolucional mencionado. El bloque convolucional mostrado a la derecha está conformado por la capa convolucional unidimensional, seguida de una capa de BatchNormalization, LeakyReLU y Dropout. Se conformo el bloque convolucional de esta forma para limitar el overfitting, que fue un problema recurrente durante el entrenamiento, y para evitar comportamientos específicos como bajas repentinas en la exactitud y subidas repentinas en la función de costo. Luego, se pasan los resultados sobre dos capas de pooling con el objetivo de extraer los elementos más importantes en el entrenamiento. Finalmente, antes de la capa de salida se envían los datos a una capa densa para filtrar mejor las características. La optimización

de hiperparámetros ajustó el número de filtros de la capa convolucional, en tamaño del kernel y la regularización (dropout y kernel). Nuevamente, para cada capa se optó por una función de activación ‘relu’ debido a que proporcionó los mejores resultados, y en la capa de salida se utilizó la función de activación ‘softmax’.

Para el caso de la CNN-LSTM, se siguió un proceso bastante similar al mencionado. Lo único que cambia de acuerdo a la arquitectura presentada del 1D-CNN, es la adición de capas de LSTM para una mejor extracción de características previas a la salida de la red. La arquitectura del modelo puede verse en la figura 11.

Figura 11. Arquitectura del modelo 1D-CNN-LSTM.

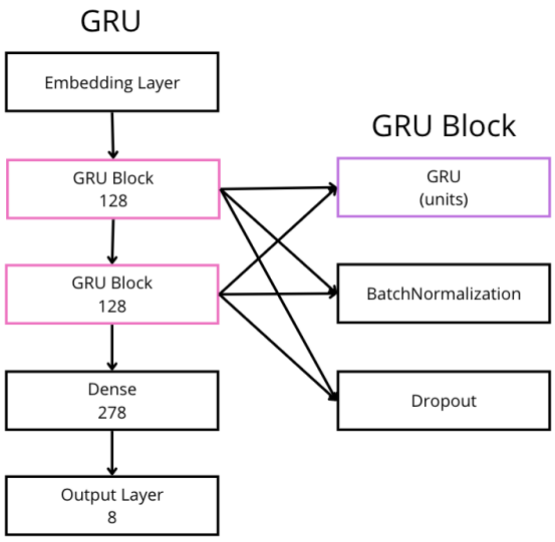


Como se puede notar en la figura 11, se sigue la misma arquitectura de la 1D-CNN, incluso utilizando la misma estructura de bloque convolucional mostrado en la figura 10. La salida del bloque convolucional está conectada a una capa LSTM, seguida del GlobalMaxPooling habitual, y las capas de salida con activaciones ‘relu’ y ‘softmax’ que ya fueron explicadas previamente.

GRU (Gated Recurrent Unit) y Transfer Learning GRU

Para el modelo siguiente, se buscó una alternativa directa a la LSTM tradicional. De aquí, se optó por un modelo de GRU, que funciona de manera similar a la LSTM ya que pertenecen a la misma familia de redes neuronales recurrentes (RNNs) para análisis de secuencias y texto. Nuevamente, se realiza el preprocesamiento habitual mencionado en la figura 8, y se implementa el modelo luego de separar el dataset preprocesado en sus conjuntos de entrenamiento, prueba y validación. La arquitectura del modelo GRU puede evidenciarse en la figura 12.

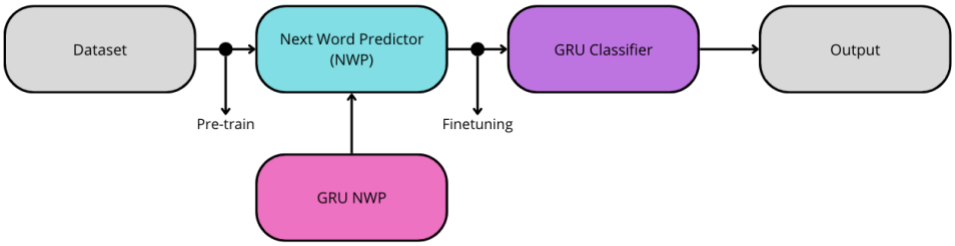
Figura 12. Arquitectura del modelo GRU.



Como se muestra en la figura 12, se empieza con una capa de Embedding, similar al acercamiento en el modelo convolucional. Luego se designan dos bloques GRU, que consisten en la capa de GRU como tal, una capa de normalización y una capa de dropout. Nuevamente, la arquitectura presentada es la que retornó los mejores resultados luego de continuas pruebas. Una vez atravesados los bloques GRU, se filtran los resultados en una capa densa con activación ‘relu’, y en la capa de salida se retorna el resultado final, a través de una función de activación ‘softmax’. La optimización se enfocó en el número de unidades en las capas de GRU, la regulariazción por dropout y el número de capas ocultas que representan el número de bloques GRU.

El último modelo que se implementó fue una experimentación. Con esto se hace referencia a que fue realizado con el propósito de explorar alternativas poco usuales. Lo que se realizó fue un modelo de predicción de siguiente palabra (Next Word Predictor) utilizando una GRU. Luego, el entrenamiento y los pesos de ese modelo entrenado fueron guardados y cargados en el modelo de GRU que se puede visualizar en la figura 12. El razonamiento detrás de esta implementación fue que un modelo ya entrenado, podría colaborar en la tarea de clasificación, facilitando la tarea inicializando pesos ya establecidos previamente. Esta técnica es conocida como 'transfer learning', en donde se utiliza un modelo pre-entrenado y se siguen métodos de 'finetuning' para entrenar un modelo diferente. El esquema de la figura 13 expresa mejor la idea de la implementación:

Figura 13. Metodología de implementación de transfer learning.



Evaluación de modelos y comparación de resultados

Antes de evaluar los modelos, es importante definir un mínimo de precisión para establecer el mínimo valor que deben superar para ser considerados como aprobados. En problemas de clasificación, lo común es determinar este mínimo utilizando la siguiente fórmula:

$$\boldsymbol{MinAcc = \frac{1}{N}}$$

donde N es el número de clases. Para clasificación binaria, al ser únicamente dos clases, el mínimo de precisión sería 0.5. Para clasificación multiclase, al tener ocho posibles clases, el valor de N sería 8, lo que establece el mínimo de precisión como:

$$\boldsymbol{MinAcc\ (8) = \frac{1}{8} = 0.125}$$

Una vez definido el límite inferior, lo que se busca maximizar es la exactitud del modelo, sin comprometer al conjunto de entrenamiento. Es decir que, los valores de exactitud entre el conjunto de entrenamiento y conjunto de prueba deben ser cercanos. Si existe una diferencia significativa, entonces se presentarían síntomas de overfitting, los cuales suceden cuando el modelo se ajusta únicamente a los datos de entrenamiento, y al analizar nuevos datos no es tan robusto ni preciso.

Sin embargo, la medida de exactitud puede no reflejar por completo la eficiencia del modelo. Por ende, nos ayudaremos de métricas adicionales basadas en el F1-score:

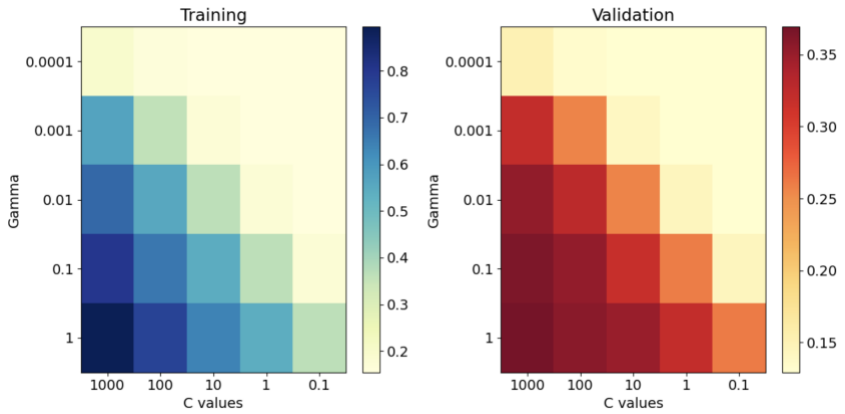
$$F1\ Score = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Debido a que este solo funciona para clasificación binaria, las alternativas a considerar son el Macro F1-score, Micro F1-score y Weighted F1-score.

Modelo base: SVM

Los primeros resultados a analizar antes de determinar la precisión final del modelo, son los resultados obtenidos durante la optimización de hiperparámetros. Se presentaron combinaciones de hiperparámetros en donde el overfitting estaba demasiado presente. Logrando una precisión en el entrenamiento de 0.96, pero una precisión en el conjunto de prueba de tan solo 0.5. Por ende, esos hiperparámetros no contribuyen a nuestro objetivo, el cual es maximizar la precisión sin comprometer el conjunto de entrenamiento.

Figura 14. Comparación entre la precisión de los conjuntos de entrenamiento y validación para optimización de hiperparámetros.



La figura 14 muestra las combinaciones de hiperparámetros, y la precisión que representan, mostrando varios casos en donde el overfitting está bastante presente. Se utilizó este gráfico para encontrar la mejor combinación que se ajuste al objetivo.

De esta forma, se optó por dos combinaciones más que producían resultados de menor precisión, pero con significativamente menos overfitting. Estos resultados se resumen en: 0.68 en el conjunto de entrenamiento y 0.42 en el conjunto de prueba para un caso que aún tiene un overfitting significativo, pero no tan exagerado; y la otra combinación que no tiene ese problema, mostrando 0.32 en el conjunto de entrenamiento y 0.29 en el conjunto de prueba.

LSTM (Long Short-Term Memory) y LSTM Bidireccional

Para el modelo de LSTM optimizado, se lograron resultados bastante prometedores. En comparación al modelo propuesto por Catak (Catak, 2020), la exactitud del modelo y los resultados de las variantes de F1 son mayores. Precisamente, en exactitud se lograron valores de alrededor del 65%, en comparación al 45% propuesto por Catak. Esto indica que el modelo de este proyecto promete bastante y valdría la pena seguir investigándolo. Sin embargo, de igual manera existe la presencia de overfitting constante durante el entrenamiento, logrando valores de alrededor del 87% en exactitud. No es tan significativo como en el modelo anterior, pero existe esa presencia. En

cuanto a los valores de F1, en macro se logró un valor de 68%, en micro se logró un 66% y en weighted le logró un 67%.

El modelo bidireccional sorpresivamente logró resultados inferiores a la LSTM unidireccional. Este modelo logró un máximo de 61% en exactitud del conjunto de prueba. Es interesante este resultado, porque se supone que los modelos bidireccionales entienden de mejor manera el contexto de oraciones. Aquí puede estar la cuestión; debido a que son secuencias, puede que no haya detectado un patrón muy efectivo y se confundió más al momento de entrenarse, logrando menores resultados.

1D-CNN (One-dimensional Convolutional Neural Network) y 1D-CNN con LSTM

Para el modelo convolucional, y su variante con las capas de LSTM en la salida, se lograron resultados prometedores también, pero no tan efectivos como la pura red LSTM. En cuanto a exactitud del modelo puramente convolucional, se logró un valor de 56% en el conjunto de prueba, y 82% en el conjunto de entrenamiento. Este modelo fue incluso más susceptible a overfitting que el anterior. Se lograron resultados de F1 de 54% en macro, 53% en micro y 53% en weighted.

Para el modelo de CNN-LSTM se lograron mejores resultados, por lo que la salida con LSTM contribuyó de manera positiva, pero fue incluso más susceptible a overfitting. Se lograron valores de exactitud de 87% en entrenamiento y 59% en prueba. En cuanto a macro F1 se logró 61%, micro F1 logró 60% y weighted F1 también logró 60%.

GRU (Gated Recurrent Unit) y Transfer Learning GRU

Para el modelo de GRU se obtuvieron resultados similares a los anteriores. Susceptible a overfitting pero un poco menos que los modelos convolucionales. La exactitud en el conjunto de entrenamiento logró un valor de 79%, y en el conjunto de prueba un valor de 58%. Para macro F1 se logró un 60%, micro F1 logró un 58% y weighted F1 logró un 59%.

Para este caso, el modelo de transfer learning superó al modelo de GRU normal. Logrando un valor de exactitud en el conjunto de prueba del 61%. Lo que demuestra que el modelo pre-entrenado si contribuyó de alguna manera al entrenamiento del clasificador, aunque no de manera muy significativa. En cuanto a resultados de F1, logró valores de alrededor del 60%, igualando al de la GRU sola.

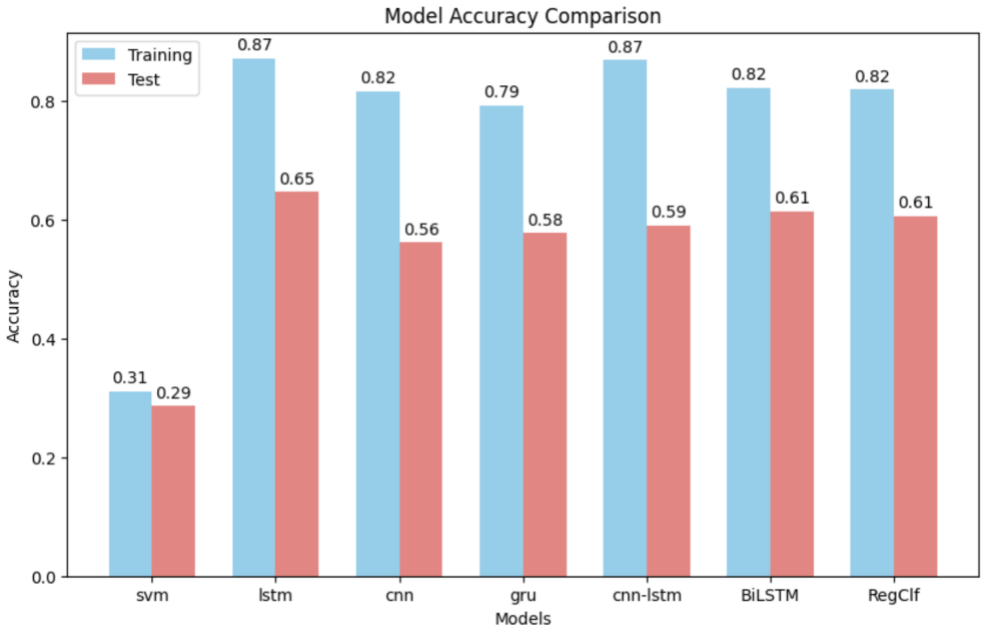
Los resultados acumulados pueden resumirse en la tabla 1 y en las figuras 15 y 16:

Tabla 1. Resumen de resultados obtenidos.

Modelo	**Entrenamiento**	**Prueba**	**Macro F1**	**Micro F1**	**Weighted F1**
SVM	0.3102	0.2862	0.2717	0.2649	0.2431
LSTM	0.8711	0.6466	0.6820	0.6637	0.6681
1D-CNN	0.8155	0.5611	0.5401	0.5250	0.5297
CNN-LSTM	0.8680	0.5898	0.6119	0.6025	0.6020
GRU	0.7921	0.5774	0.6015	0.5838	0.5926
Bidirectional LSTM	0.8212	0.6134	0.6345	0.6313	0.6297
Transfer Learning	0.8181	0.6055	0.6008	0.5850	0.5875

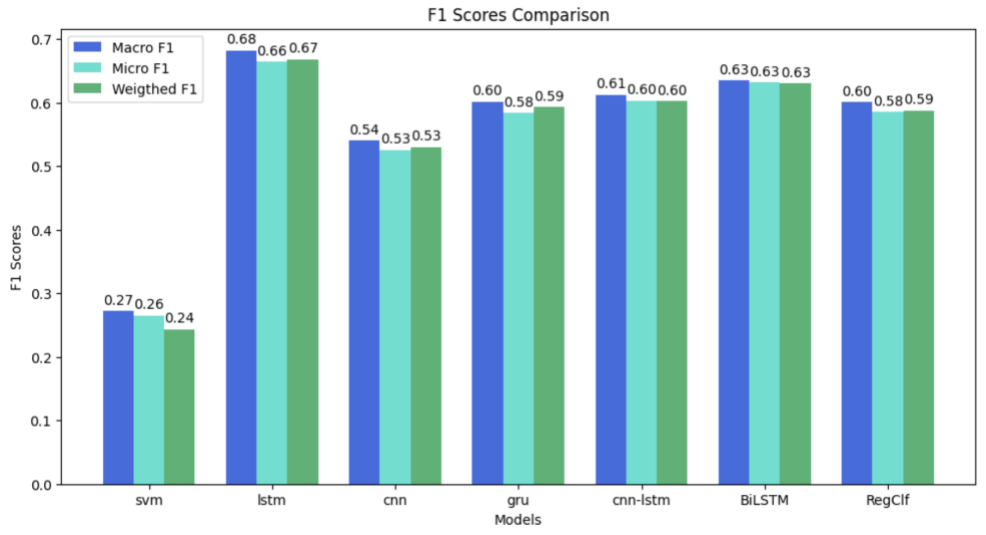
La figura 15 expresa de mejor manera los resultados obtenidos para la exactitud en el entrenamiento y prueba:

Figura 15. Resumen de resultados para entrenamiento y prueba.



La figura 16 muestra los resultados obtenidos según las variantes de F1 para problemas multiclase:

Figura 16. Resultados de Macro F1, Micro F1 y Weighted F1.



Análisis estadístico y pruebas de hipótesis

A simple vista, con los resultados obtenidos y mostrados en los gráficos anteriores, se puede notar a simple vista que hay un modelo que destaca por encima del resto: LSTM. Sin embargo, no se

podría decir que estadísticamente es el mejor modelo de todos sin antes realizar una prueba de hipótesis que especifique si existe o no diferencia significativa entre dos modelos. Para las pruebas de hipótesis se utilizarán valores obtenidos de un proceso conocido como validación cruzada. Esta consiste en entrenar el modelo dividiendo el dataset por intervalos (folds), y anotando resultados del entrenamiento por cada división. De esta validación cruzada se pueden obtener valores promedio, desviación estándar y valores individuales necesarios para la prueba de hipótesis.

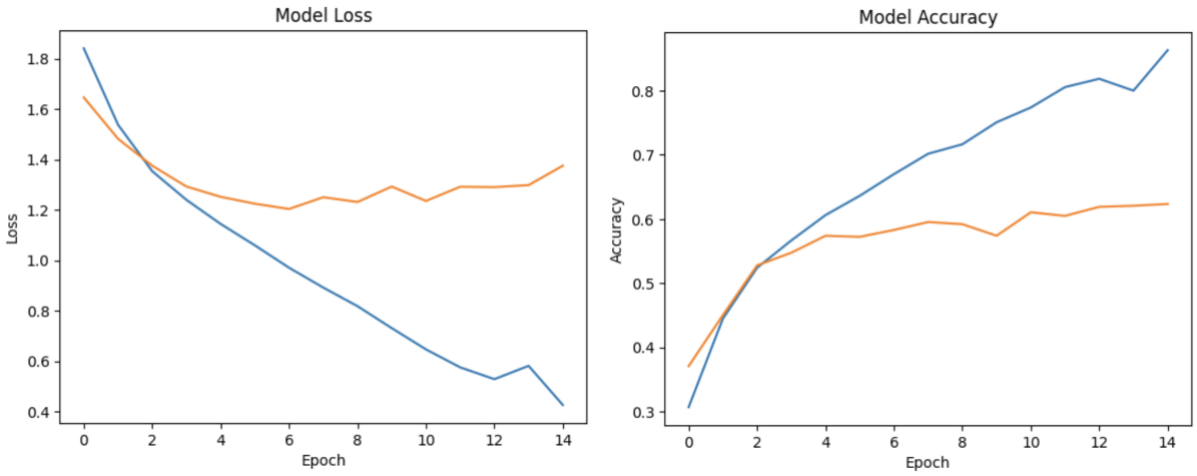
Una prueba de hipótesis es un proceso común en el campo de la estadística. Consiste en definir una hipótesis nula (H0) y una hipótesis alternativa (H1). Se define un valor de significancia alpha (α); para este caso $\alpha = 0.05$. Se recopilan los datos (obtenidos de la validación cruzada mencionada previamente) y se define una métrica asociada a los datos. Para este caso, se utilizarán los valores obtenidos por la métrica 'Weighted F1-score'. Debido a que el dataset está un poco desbalanceado ('weighted F1-score toma en cuenta el desequilibrio en las clases), se cree que esta métrica es la más efectiva para determinar la potencia de los modelos. Una vez recopilados los datos y seleccionada la métrica, se realiza la prueba de hipótesis. Para este caso, se utilizó Wilcoxon para determinar si existe diferencia significativa en las muestras de datos. Si el resultado de la prueba de Wilcoxon retorna un resultado menor que alpha, entonces se rechaza H0 y se acepta H1. Esto significa que se determina que un modelo es estadísticamente mejor sobre otro.

Se realizó la prueba de Wilcoxon para el modelo de LSTM, contra todos los demás modelos. Los resultados obtenidos demostraron que el modelo LSTM es significativamente mejor que los demás, rechazando las hipótesis nulas. Al obtener un valor-p de alrededor de 0.00195 se aceptan las hipótesis alternativas, que muestran a la LSTM como claro vencedor sobre las demás.

Figuras y métricas adicionales del modelo de mejor rendimiento

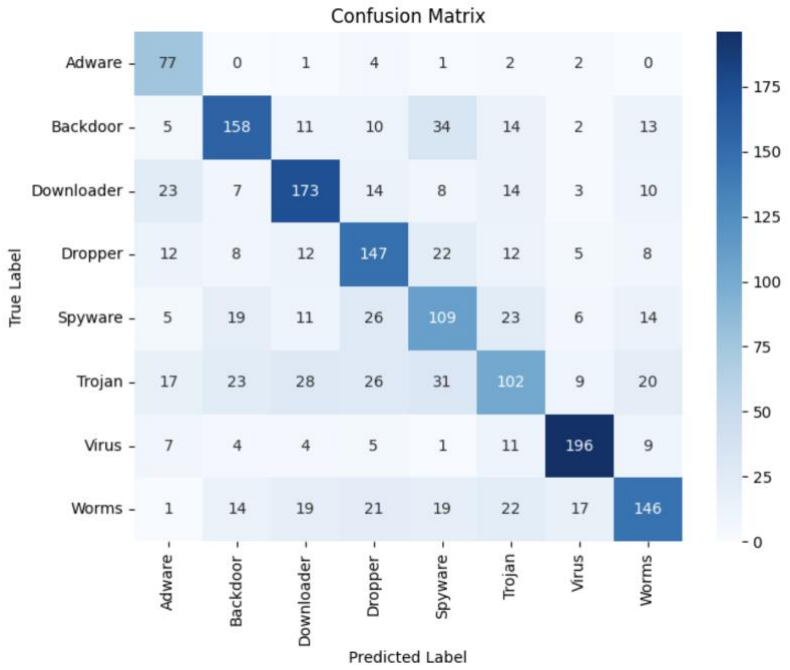
Esta sección tiene como objetivo mostrar algunos gráficos interesantes realizados durante la implementación. Además, de mostrar métricas adicionales que pueden ser relevantes para la comprensión del comportamiento del modelo.

Figura 17. Gráficos de la función de costo y exactitud del modelo por cada época de entrenamiento.



La figura 17 es sumamente importante en proyectos de machine learning, ya que puede mostrar comportamientos acerca del modelo como overfitting, underfitting y demás. En este caso, el modelo sufre de overfitting. Esto se sabe por el comportamiento de la función de costo (izquierda). La curva de costo del entrenamiento disminuye, pero la curva de prueba comienza a ser constante y elevarse un poco. Por la naturaleza del dataset, el overfitting es un problema común en todos los modelos implementados. Aunque si hay técnicas para combatir este comportamiento, se decidió tratar de maximizar la exactitud en el conjunto de prueba, permitiendo un comportamiento considerable de overftting. Este problema describe a los modelos como “poco robustos”, y limitados al momento de obtener datos y resultados reales. Solo está ajustado al conjunto de entrenamiento.

Figura 18. Matriz de confusión del modelo LSTM.



La figura 18 es una matriz de confusión, una herramienta común en el análisis de modelos de clasificación. Expresa de manera gráfica las predicciones del modelo con las clases reales. Permite realizar interpretaciones como proporciones entre verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. De estos valores se pueden obtener tablas como la siguiente:

Tabla 2. Reporte de clasificación para el modelo LSTM.

Clase	Precisión	Recall	F1-Score	Muestras
Adware	0.52	0.89	0.66	87
Backdoor	0.68	0.64	0.66	247
Downloader	0.67	0.69	0.68	252
Dropper	0.58	0.65	0.61	226
Spyware	0.48	0.51	0.50	213
Trojan	0.51	0.40	0.45	256
Virus	0.82	0.83	0.82	237
Worms	0.66	0.56	0.61	259

La tabla 2 muestra los resultados del reporte de clasificación para el modelo. Para resumir un poco, la tabla presenta como se comportan las proporciones para cada clase, utilizando métricas como la precisión y el ‘recall’, las cuales son medidas de proporción entre falsos positivos, verdaderos positivos y verdaderos negativos.

CONCLUSIONES

Este estudio resalta el papel crítico de la ciberseguridad en la preservación de la integridad de la información a nivel mundial en medio del cambiante panorama digital. La creciente sofisticación de las amenazas cibernéticas, ejemplificada particularmente por el malware metamórfico, exige enfoques avanzados y automatizados para la identificación y clasificación. El aprovechamiento del aprendizaje automático, específicamente las redes neuronales recurrentes (RNNs) y las redes neuronales convolucionales unidimensionales (1D-CNNs), ha demostrado ser fundamental para analizar y categorizar estas amenazas evasivas.

El análisis exhaustivo de los comportamientos del malware metamórfico, utilizando un conjunto de datos que presenta llamadas secuenciales al API de Windows, resultó en la clasificación de ocho tipos distintos: spyware, downloader, troyanos, gusanos, adware, dropper, virus y backdoors. La aplicación de técnicas de aprendizaje automático, que abarcan el preprocesamiento de datos, visualización, implementación de modelos y análisis estadísticos, facilitó una evaluación comparativa de los modelos. Destacadamente, el modelo LSTM se posicionó como el más efectivo, logrando un 65% de precisión y aproximadamente un 68% en clasificación multiclase.

La relevancia de este trabajo radica en su contribución a la detección y neutralización efectivas de malware mediante enfoques prometedores de aprendizaje automático. La importancia se extiende más allá de los hallazgos inmediatos, enfatizando el desarrollo continuo en el campo de la ciberseguridad.

En reflexión, el estudio destaca la importancia de la representación gráfica para comunicar los resultados de manera más efectiva. La dominancia del modelo LSTM, con otros alcanzando

niveles de precisión entre el 50% y 60%, excluyendo el modelo base, subraya la eficacia del enfoque seleccionado.

Se recomienda que estudios futuros exploren metodologías alternativas de preprocesamiento, considerando el desafío planteado por comportamientos similares en secuencias analizadas en diferentes clases. La exploración de diferentes metodologías de optimización para modelos y la limpieza del conjunto de datos y preprocesamiento son caminos cruciales para investigaciones adicionales.

La investigación enfrentó desafíos en recursos computacionales, principalmente en términos de limitaciones de RAM y acceso a GPU, subrayando la necesidad de una infraestructura mejorada en futuros esfuerzos. Otro desafío común fue la constante aparición de comportamientos de *overfitting* en los modelos, y la recurrencia en la optimización de los modelos para tratar de minimizar estos problemas. A pesar de estos desafíos, el estudio proporciona perspicacias valiosas para combatir malware avanzado mediante aprendizaje automático, contribuyendo al desarrollo continuo de prácticas de ciberseguridad.

REFERENCIAS BIBLIOGRÁFICAS

Akhtar, Muhammad Shoaib & Feng, Tao. (2022). Detection of Malware by Deep Learning as CNN-LSTM Machine Learning Techniques in Real Time. Symmetry. 14. 2308. 10.3390/sym14112308. Obtenido de: https://www.researchgate.net/publication/365140194_Detection_of_Malware_by_Deep_Learning_as_CNN-LSTM_Machine_Learning_Techniques_in_Real_Time?_tp=eyJjb250ZXh0Ijp7InBhZ2UiOiJwdWJsaWNhdGlvbiIsInByZXZpb3VzUGFnZSI6bnVsbH19

Akinsola, Osisanwo, Awodele, Hinmikaiye, Olakanmi & Akinjobi (2017). Supervised Machine Learning Algorithms: Classification and Comparison. 2017. International Journal of Computer Trends and Technology (IJCTT), Vol. 48, Number 3. Babcock University, Nigeria. Obtenido de: https://www.researchgate.net/profile/J-E-T-Akinsola/publication/318338750_Supervised_Machine_Learning_Algorithms_Classification_and_Comparison/links/596481dd0f7e9b819497e265/Supervised-Machine-Learning-Algorithms-Classification-and-Comparison.pdf

Brezinski, Kenneth & Ferens, Ken. (2023). Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits. Security and Communication Networks. 2023. 8227751. 10.1155/2023/8227751. Obtenido de: https://www.researchgate.net/publication/373639556_Metamorphic_Malware_and_Obfuscation_A_Survey_of_Techniques_Variants_and_Generation_Kits

Catak, Ferhat Ozgur & Yazi, Ahmet. (2019). A Benchmark API Call Dataset for Windows PE Malware Classification. Obtenido de: https://www.researchgate.net/publication/332877263_A_Benchmark_API_Call_Dataset_For_Windows_PE_Malware_Classification

Catak, Ferhat Ozgur & Yazi, Ahmet & Elezaj, Ogerta & Ahmed, Javed. (2020). Deep learning based Sequential model for malware analysis using Windows exe API Calls. PeerJ Computer Science. 6. 10.7717/peerj-cs.285. Obtenido de: https://www.researchgate.net/publication/331974598_Classification_of_Methamorphic_Malware_with_Deep_Learning_LSTM

Graves, A., Jaitly, N., & Mohamed, A. R. (2013, December). Hybrid speech recognition with deep bidirectional LSTM. In *2013 IEEE workshop on automatic speech recognition and understanding* (pp. 273-278). IEEE. Obtenido de: https://www.cs.toronto.edu/~graves/asru_2013.pdf

Le, Boydell, Mac Namee & Scanlon. (2018). Deep learning at the shallow end: Malware classification for non-domain experts. DFRWS 2018 USA, obtenido de: https://arxiv.org/ftp/arxiv/papers/1807/1807.08265.pdf

Staudemeyer, R. C., & Morris, E. R. (2019, September 23). Understanding LSTM, a tutorial into Long Short-Term Memory Recurrent Neural Networks. Obtenido de: https://arxiv.org/pdf/1909.09586.pdf

Tianliang Lu, Yanhui Du, Li Ouyang, Qiuyu Chen, Xirui Wang, "Android Malware Detection Based on a Hybrid Deep Learning Model", *Security and Communication Networks*, vol. 2020, Article ID 8863617, 11 pages, 2020. https://doi.org/10.1155/2020/8863617