# Microsoft Malware Detection

## Proyecto 2

Michele Benvenuto [a], Roberto Figueroa [b], Gustavo Méndez[c]

Departamento de Ing. en Ciencias de la Computación y T.I

Universidad del Valle de Guatemala, Guatemala

[a] ben18232@uvg.edu.gt, [b]fig18306@uvg.edu.gt, [c]men18500@uvg.edu.gt

**Resumen— Este documento describe el proceso de implementación de modelos de aprendizaje supervisado en un dataset proporcionado por Microsoft, donde los datos provienen de computadoras personales, de negocios, etc. Los modelos implementados son los siguientes: *Naive Bayes, MLP Classifier y XGBoost.* Tras realizar la exploración, derivación y selección de características, síntesis de características y preprocesamiento, los modelos se sometieron al mismo dataset de entrenamiento. El modelo que presentó las mejores métricas fue el XGBoost que obtuvo una precisión de 0.6495.**

**Abstract— This document describes the process of implementing supervised learning models on a data set provided by Microsoft, where the data comes from personal computers, business computers, etc. The implemented models are the following: NaiveBayes, MLP Classifier and XGBoost. After performing exploration, feature derivation and selection, feature synthesis, and preprocessing, the models were subjected to the same training dataset. The model with the best performance was the XGBoost with a precision of 0.6495**

***Palabras clave—seguridad, ciberseguridad, ataques, detección, ciencia de datos, malware, Microsoft, Naive Bayes, SVC, XGBoost.***

## I. Introducción

La ciberseguridad es el principal desafío de nuestra era digital. La industria del malware sigue siendo un mercado bien organizado y bien financiado dedicado a evadir las medidas de seguridad tradicionales. Una vez que una computadora se infecta con malware, los delincuentes pueden dañar a los consumidores y las empresas de muchas maneras [3]. En un mundo interconectado, las personas se mantienen productivas y conectadas mediante una variedad de dispositivos. Si bien hay un valor increíble en tanta conectividad y productividad, hay un crecimiento correspondiente en el riesgo a medida que las personas aumentan su exposición a las amenazas de seguridad cibernética.

¿Por qué es entonces importante que las soluciones actuales antimalware sean prioritarias? Todos los productos de protección buscan la perfección: todos quieren clientes que nunca se infecten y que nunca experimenten detecciones incorrectas (falsos positivos). Sin embargo, la era de combatir el malware con firmas estáticas y precisas ya es cuestión del pasado. Los proveedores de antimalware deben aprovechar las metodologías, la automatización y el aprendizaje automático de última generación para combatir las amenazas a las que se pueden enfrentar los usuarios.

Microsoft ha estado invirtiendo mucho en tecnologías de seguridad de última generación. Estas tecnologías utilizan nuestra capacidad para consolidar grandes conjuntos de datos y construir sistemas inteligentes que aprenden de esos datos. *Machine Learning* ha mejorado la forma en que *Windows Defender Advanced Threat Protection (Windows Defender ATP)* detecta ataques avanzados, incluidas las actividades de los atacantes principales que normalmente residen solo en la memoria o se camuflan como eventos desencadenados por herramientas comunes y aplicaciones cotidianas.

Al tener en cuenta miles de señales o metadatos de las interacciones de las aplicaciones, ML puede dividir los datos con mayor precisión mientras se guía por heurísticas creadas manualmente. Según el análisis de alertas reales, las tecnologías de ML implementadas en Microsoft son al menos un 20% más precisas que las heurísticas creadas manualmente [10]. Si bien el software de seguridad mejora sus enfoques analíticos mediante la aplicación de aprendizaje automático (ML), los ataques que violan estos análisis automáticos se reproducen exponencialmente. El crecimiento constante de malwares nuevos y más complejos hace que sea imposible detectar todos estos posibles ataques juntos.

Con las herramientas de *Machine Learning* de código abierto disponibles en la actualidad, se pueden crear herramientas personalizadas de detección de malware basadas en aprendizaje automático, ya sea como su herramienta de detección principal o para complementar soluciones comerciales, con relativamente poco esfuerzo. En este artículo se describe la manera en cómo herramientas como Scikit Learn (librería de Python) pueden ayudar a la detección y clasificación de malware mediante la creación de modelos de aprendizaje supervisado.

## II. Marco Teórico

### Malware

El término malware proviene de la abreviación de software malicioso y puede tener muchas finalidades como comprometer equipos de cómputo, acceder a bases de datos o extraer información relevante. Un malware puede infectar un dispositivo de manera inmediata, y es por ello que se sigue buscando nuevas y mejoradas maneras de poder analizar y clasificar malwares [7].

Su análisis comprende diferentes enfoques, mientras que en la práctica se emplean técnicas para obtener información más precisa. Entre los tipos de análisis, se puede resaltar dos tipos:

- Análisis estático: este tipo de análisis no ejecuta el malware, en cambio, se hace un análisis de código. La información individual o conjunta hace posible la detección del malware, extrayendo características para evaluar el impacto que tienen al momento de clasificarlo como tal.
- Análisis dinámico: a diferencia del estático, implica el poder analizar la ejecución del malware y cómo repercute en el entorno seleccionado. Generalmente se elige un *sandbox* de pruebas para obtener información del malware de manera segura, al observar su comportamiento en tiempo de ejecución.

### Acerca de la competencia

Microsoft, en colaboración con *Windows Defender ATP Research, Northeastern University College of Computer and Information Science,* y *Georgia Tech Institute for Information Security & Privacy* han propuesto en el año 2019 una competición en la plataforma Kaggle a la comunidad de *Data Science* para desarrollar técnicas para predecir si una máquina pronto se verá afectada por malware. El objetivo de esta competencia consiste en poder predecir la probabilidad de que una máquina con Windows se infecte con varias familias de malware, en función de las diferentes propiedades de esa máquina [3]. Los datos de telemetría que contienen estas propiedades y las infecciones de la máquina se generaron mediante la combinación de informes y amenazas recopilados por Windows Defender, producto estrella de Microsoft para poder combatir amenazar y mantener a los dispositivos seguros. Los participantes tienen como reto el construir modelos utilizando 9,4 GB de datos anónimos de 16,8 millones de dispositivos, y los modelos resultantes se calificarán según su capacidad para hacer predicciones correctas. Los equipos ganadores obtienen $25,000 en premios totales [10].

Si bien la seguridad siempre ha sido una prioridad para Microsoft, este nuevo mundo requiere un nuevo enfoque y una gran inversión en prevención, detección y respuesta a amenazas. Windows Defender, junto con muchas otras funciones integradas en Windows 10, están en primera línea y deben de mejorar cada vez ante los riesgos del futuro. Es justo aquí donde temas como *Data Science* unen fuerzas para la detección de amenazas, incluso en tiempo real.

### Windows Defender

Windows Defender tiene un sistema de protección que puede ser soportado en la nube, donde suele emitir un veredicto en milisegundos basándose únicamente en metadatos. Si este análisis de nubes ligeras es insuficiente para llegar a una determinación, se solicita la muestra para un análisis más profundo y un procesamiento posterior. Este análisis profundo completamente automatizado brinda evaluaciones precisas a los usuarios, pero también tiene la ventaja de brindar protección inmediata contra amenazas similares en dispositivos de todo el mundo en función del análisis automatizado de esa muestra. Este proceso de análisis y la protección que genera tardan solo unos segundos en completarse [8].

De acuerdo a Microsoft [6], se procesan alrededor de 4.5 millones de archivos diarios para poder analizar información clave de malware. Los clientes de Windows Defender experimentan alrededor de 90 mil millones de encuentros potencialmente maliciosos por día que necesitan un veredicto. En un día cualquiera, alrededor del 97% de estos veredictos los hace el cliente. El 3% restante de estos encuentros, alrededor de 2 a 3 mil millones de consultas por día, son procesados por el sistema de protección en la nube de Windows Defender.

### Machine Learning como complemento, no respuesta

En otras empresas como Kaspersky, también han dedicado un esfuerzo por implementar herramientas de aprendizaje en sus productos. Entre ellos destacan los conjuntos de árboles de decisión, el hashing sensible a la localidad, los modelos de comportamiento o la agrupación de flujo entrante, diseñados para cumplir con los requisitos de seguridad del mundo real: baja tasa de falsos positivos, interpretabilidad y robustez para un adversario potencial [4]. Sin embargo, aún hay retos por resolver en este mundo donde la seguridad y el valor de los datos juegan un papel muy importante.

Un atacante puede envenenar un conjunto de datos de entrenamiento o aplicar ingeniería inversa al código del modelo. Además, los piratas informáticos pueden usar modelos de ML de "fuerza bruta" con agentes de IA (adversarios) especialmente desarrollados para generar automáticamente muchas muestras de ataque hasta que se descubra un punto débil del modelo. Por tanto, las empresas como Microsoft y Kaspersky deberían de poder comprender y abordar cuidadosamente los requisitos esenciales para el

rendimiento de *Machine Learning* en el mundo real, potencialmente hostil y ofrecer solidez frente a posibles atacantes o adversarios.

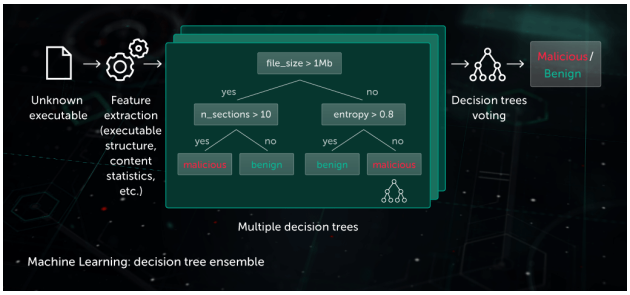


*Figura 1.* Pipeline de herramientas de ML en Kaspersky

Por último, un modelo entrenado para combatir malware no debe verse como la respuesta definitiva al problema, ya que en realidad debe ser parte de un enfoque de seguridad multicapa, donde las tecnologías complementarias de ML trabajen en conjunto con la experiencia humana: trabajando para ofrecer a sus usuarios una mejor protección contra las amenazas actuales y las que surgirán el día de mañana.

## III. Propuesta de metodología

Se contó con dos conjuntos de datos los cuales uno era de entrenamiento y otro era para pruebas. El primero contaba con la etiqueta *HasDetections* que clasificaba a dicha observación como una entidad que tenía malware. Para el segundo conjunto de datos no se contaba con esta etiqueta dado que se trata de una competición. Las fases para desarrollar tres modelos de clasificación incluyen: análisis exploratorio, preprocesamiento de la data, selección de características, separación de datos de entrenamiento y pruebas, implementación de los modelos y la evaluación de los mismo a través de métricas como curva ROC, precisión, *recall*, exactitud, matriz de confusión y *F1-score.*

### A. *Análisis Exploratorio*

Durante esta fase se exploró cómo las observaciones eran registradas en el conjunto de datos de entrenamiento. Inicialmente se contaban con 8921483 observaciones y 83 columnas. La cantidad de observaciones etiquetadas con positivo a malware era simétrica a la cantidad con negativo a malware, es decir, las observaciones estaban balanceadas, ver Figura 2. Se separaron las columnas según su tipo, categoría o numérica. Con esto se procedió a explorar cada tipo logrando conocer más sobre la información que la cada columna almacenaba.

Dentro de algunos descubrimientos relevantes se encuentra que la mayor parte de los datos representan a computadoras con Windows 10. Además, la arquitectura predominante es la de 64 bits. Se observó también que la mayor parte de las entidades o equipos registrados contenían procesadores de media a alta gama. En cuanto a seguridad se observa que existe una predominancia en los equipos a tener Firewall.

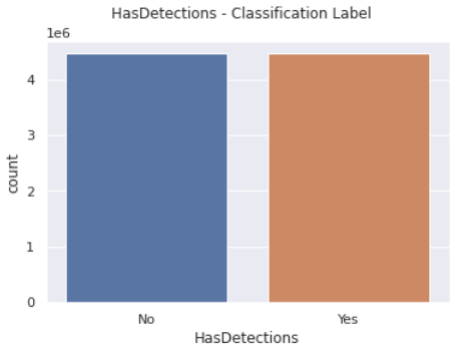


*Figura 2.* Conteo de observaciones etiquetadas positivo a detección de malware (Yes) y negativo a detección de malware (No).

En cuanto a los datos se encontró que 74 columnas tenían al menos un dato faltante. En particular el 7% de las columnas del dataset contenían más de la mitad de valores faltantes. Para esto se tomó la decisión de eliminar aquellas columnas que estaban compuestas en su mayoría por datos faltantes. En cuanto a las columnas como SmartScreen, Census_InternalBatteryType y OrganizationIdentifier contenían relativamente pocos valores faltantes por lo que se procedió a sintetizar dicha información.

### B. *Preprocesamiento*

Para esta fase se tomaron en cuenta las observaciones de la información que el análisis exploratorio pudo dilucidar.

Se comenzó por derivar características de columnas existentes con el fin de enriquecer el conjunto de datos. Las columnas y la manera que se aplicó dicha derivación se lista a continuación:

- EngineVersion: Se derivaron a partir de esta columna los números de versión menores (minor version numbers) y los números de revisión (revision number).
- AppVersion: Se procedió a derivar tres columnas a partir de esta versión aplicando la misma técnica que la característica anterior.
- Census_OSVersion: Se derivaron cuatro columnas con técnica similar a las columnas anteriores.

Se procedió a codificar todas las columnas categóricas. Con esto se dió paso a buscar columnas correlacionadas entre ellas con el fin de reducir la cantidad de columnas. Aquellas que obtuvieron una correlación superior a 75% fueron seleccionadas con el fin de eliminar una de las dos.

### C. *Selección de features*

Esta fase se hizo en paralelo con la anterior, ya que involucra seleccionar aquellas características correlacionadas al menos en un 10-20%. Las características seleccionadas se muestran en la Tabla 1. Además, se utilizó *SelectKBest* y *ExtraTreesClassifier* solo para obtener las características más importantes. Por último, se utilizó criterio de investigación para poder tomar una mejor decisión a la hora de elegir o no cierta característica.

TABLA 1. FEATURES SELECCIONADAS

| ***Feature*** | **Descripción** |
|---|---|
| SmartScreen | Es el string habilitado de SmartScreen, se obtiene de HKLM\SOFTWARE\Policies\Microsoft\Windows\System\SmartScreenEnabled and HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\SmartScreenEnabled |
| AVProductStatesIdentifier | ID para la configuración específica del antivirus del usuario |
| Census_OSArchitecture | La arquitectura en la que se basa el sistema operativo |
| AppVersion_2 | Información de estado de defender |
| EngineVersion_3 | Información de estado de defender |
| Census_TotalPhysicalRAM | La memoria ram del dispositivo en MB |
| IsProtected | Verdadero si existe un antivirus activo y con la versión actual en la máquina, Falso sí no |
| EngineVersion | Información de estado de Defender |
| Census_SystemVolumeTotalCapacity | El tamaño de la partición en la cual el sistema está instalado en MB |
| CountryIdentifier | ID del país en el cual la máquina se encuentra |
| OrganizationIdentifier | ID de la organización a la cual pertenece la máquina, tanto para compañías específicas como para más generales |
| AVProductsInstalled | NA |
| AppVersion_3 | Información de estado de defender |
| Census_FirmwareVersionIdentifier | NA |
| Census_ProcessorModelIdentifier | NA |
| Census_OEMModelIdentifier | NA |
| **HasDetections** | Variable a predecir, etiqueta que indica si un malware fue detectado en la máquina |

### D. *Separación de datos*

Debido a las dimensiones del dataset se estableció que esta fase se realizaría antes de entrenar los modelos pues de esta manera se cuenta con un dataset reducido especialmente en características.

### E. *Implementación de modelos*

Para la implementación de modelos se utilizó la librería de Python llamada *Scikit Learn*. Esta librería ofrece diversos modelos de clasificación supervisados pero para este proyecto se tomaron en cuenta *Naive Bayes,* MLP *y XGBoost (este último pertenece a otra librería, pero tiene soporte con Scikit)*. Las *features* seleccionadas en su mayoría comprenden datos numéricos, por lo que esto también fue factor para la selección de los mismos.

Se tomó en cuenta *Naive Bayes* ya que es un modelo relativamente simple que no requiere de una gran cantidad de datos para poder estimar los parámetros para realizar las clasificaciones. Los clasificadores *Naive Bayes* se basan en el teorema de Bayes suponiendo que todas los valores de las observaciones son independientes. A pesar de esta fuerte suposición estos modelos han tenido resultados sorprendentes al momento de analizar situaciones complejas [1].

También se seleccionó el modelo MLPClassifier. Hace referencia a *Multi-layer Perceptron classifier* o clasificador de perceptrones de múltiples capas. A diferencia de los clasificadores como *Support Vectors* o *Naive Bayes* este modelo realiza la tarea de clasificación por medio de capas de neuronas. Se utilizó redes neuronales ya que se quiso poner a prueba su eficacia ya que la peculiaridad de este tipo de algoritmos es su habilidad de poder encontrar relaciones con información no lineal y compleja como es nuestro caso. Además por la naturaleza del problema no se puede garantizar que todas las computadoras tengan las mismas características (como en la vida real) por lo que existe heterocedasticidad (alta volatilidad y nula constancia en la varianza de las observaciones), siendo este un factor clave que las redes neuronales pueden manejar [5].

Inicialmente una red neuronal tiene todas sus ponderaciones o pesos asignados de manera aleatoria, dando resultados de clasificación totalmente erróneos. La red aprende a través del entrenamiento. Continuamente se presentan en la red ejemplos para los que se conoce el resultado y las respuestas se comparan con los resultados conocidos, en este caso se le brindó a la red las características seleccionadas con la etiqueta de malware respectiva. La información procedente de esta comparación se pasa hacia atrás a través de la red, cambiando las ponderaciones o pesos gradualmente. A medida que progresa el entrenamiento, la red se va haciendo cada vez más precisa en la replicación de resultados conocidos [2].

La red fue definida con 100 neuronas en cada capa escondida, Adam para la optimización de pesos, la función de activación en cada neurona fue Relu (rectified linear unit function), el término de regularización o parámetro de penalización fue de 0.0001 al igual que la tasa de aprendizaje. Se estableció que el modelo detuviera su entrenamiento al observar que el puntaje de validación no mejoraba.

Por último, XGBoost ha sido utilizado ampliamente en competencias debido a su buen rendimiento. Es una biblioteca de aumento de gradiente distribuida optimizada diseñada para ser altamente eficiente, flexible y portátil. Implementa algoritmos de aprendizaje automático bajo el marco Gradient Boosting, y resuelve muchos problemas de ciencia de datos de una manera rápida y precisa [11].

### *F. Métricas*

Finalmente, las métricas básicas a obtener de estos modelos son las siguientes: matriz de confusión, *accuracy, precision, recall, F1 score*. Además, a manera de validar estos valores, se realizó el proceso de validación cruzada con *K-Folds* (con k=10) así como la graficación de la curva ROC para poder mostrar el rendimiento de los modelos de clasificación con base en todas sus clasificaciones (teniendo en cuenta los falsos positivos y los falsos negativos).

El resumen de dichas métricas puede observarse en la Tabla 2 y 3, además de poder observar las curvas ROC en los Gráficos 1, 2 y 3 para cada modelo, respectivamente.

TABLA 2. MÉTRICAS BÁSICAS POR MODELO

| | ***Naive Bayes*** | ***MLPClassifier*** | ***XGBoost*** |
|---|---|---|---|
| *Accuracy* | 0.5798 | 0.6322 | 0.6494 |
| *Precision* | 0.5940 | 0.6322 | 0.6495 |
| *Recall* | 0.5798 | 0.6322 | 0.6494 |
| *F1 score* | 0.5628 | 0.6322 | 0.6493 |

TABLA 3. MÉTRICAS AUXILIARES POR MODELO

| | *Naive Bayes* | *MLPClassifier* | *XGBoost* |
|---|---|---|---|
| *K-Fold validation* | $0.5796 \pm 0.000490$ | $0.6313 \pm 0.000529$ | $0.6934 \pm 0.000167$ |
| Área en curva ROC | 0.55 | 0.63 | 0.65 |

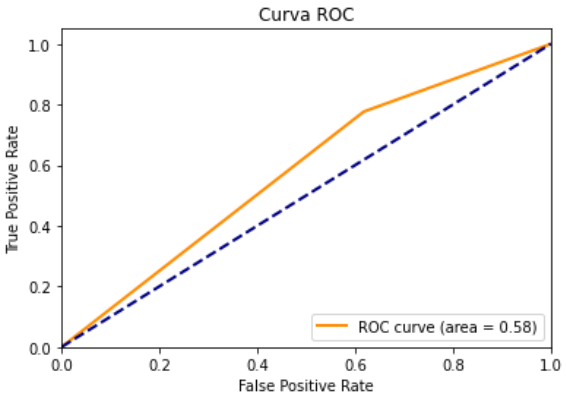


GRÁFICO 1. CURVA ROC *NAIVE BAYES*

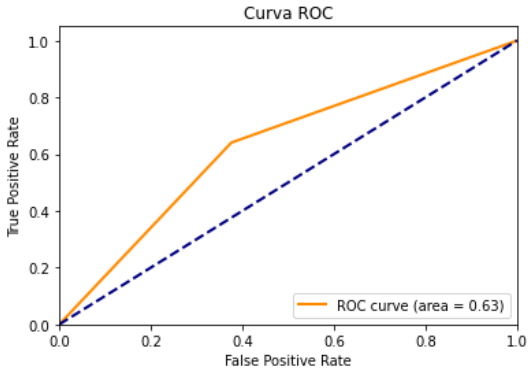


GRÁFICO 2. CURVA ROC MLPCLASSIFIER

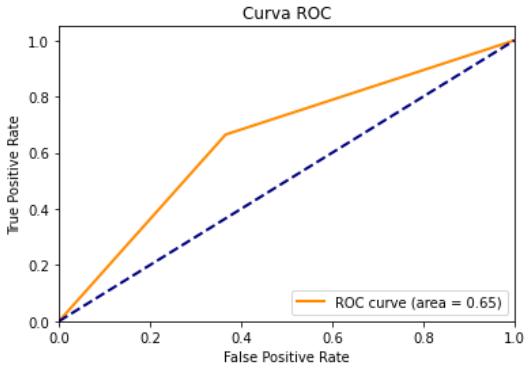


GRÁFICO 3. CURVA ROC XGBOOST CLASSIFIER

## IV. DISCUSIÓN DE RESULTADOS

Predecir que una computadora está infectada sin duda alguna es una ventaja pues permite realizar acciones preventivas o correctivas, incluso permite determinar características claves que hace susceptibles a ataques a cierto equipo de cómputo. Inicialmente se tenía un conjunto de datos que tenía 83 features por cada observación. Se llegó a reducir un 75% de esas características con el fin de poder tener una cantidad manejable y que además los modelos se pudieran entrenar con mayor facilidad. Otra ventaja de reducir la cantidad de columnas fue que se pudo observar características que están correlacionadas con que a una computadora se le detecte malware, como se puede observar en la Tabla 1. Esta tabla nos muestra características que pueden ser obvias como que el equipo esté protegido o no o el estado del SmartScreen.

Sin embargo, otras características nos llegan a dar más información acerca de los atacantes y sus intereses como la organización a la que pertenece el equipo o el lugar donde este se encuentre. Incluso existen características que llegan a brindar información acerca de vulnerabilidades propias de Windows como son las versiones del software. Otra peculiaridad es poder observar las características del hardware, probablemente esto sea factor para los atacantes pues al observar que se cuenta con mínimos requerimientos se podría tratar de una máquina virtual con propósitos diferentes a los de una computadora para jugadores de videojuegos.

En cuanto a los modelos se observa que a pesar de reducir la cantidad de columnas para evitar ruido, estos no logran manejar niveles altos de clasificación. Esto puede ser por la poca linealidad que manejan los datos y también por su complejidad, lo cual es amplio tema para una discusión. Incluso el escalar los datos de manera estándar no aportó tanto cambio para las métricas de los modelos. Los modelos suelen comportarse de manera no óptima y esto es debido a diversos factores y a la manera, algunas veces estocástica, que operan. Las limitaciones que se encuentran al entrenar modelos pueden ser por la interpretabilidad, la calidad de la información y la complejidad de lo que queremos modelar o clasificar [9].

La interpretabilidad en este caso nos dice que debemos tener conciencia de cómo opera el modelo y no podemos solo dejarlo a las manos del algoritmo (como si fuera una caja negra), en este caso debemos reconocer qué variables son las que mayor guardan relación con que una máquina pueda estar infectada y cómo funciona el algoritmo para tomar sus decisiones. En cuanto a la calidad de la información, el *dataset* proveído puede que no contenga toda la información que represente el estado de una computadora, debido a factores legales y de privacidad, por lo que puede llegar a ser pobre y esto hace que los modelos no puedan llegar a determinar con claridad una correcta clasificación. La complejidad es inherente al problema, en este caso el problema de clasificar ya supone un reto y la entropía con que se encuentra en el ámbito de malware es alta, sobre todo porque no todos tenemos la misma computadora ni los mismo hábitos de uso.

En relación a los puntajes de los modelos se puede observar que las métricas para el modelo basado en *Naive Bayes* tienden a estar en los rangos de 0.56 - 0.59 siendo el modelo con peor desempeño. El bajo rendimiento de este modelo en

comparación a los otros dos modelos probablemente se debe a que el modelo asume que las variables de las observaciones son independientes lo cual es poco probable en situaciones complejas como la que se presenta en este proyecto. A pesar de esto las ventajas que presenta este modelo es que fue el modelo que entrenó con mayor velocidad y las métricas no son considerablemente menores en comparación a los otros modelos, la mayoría de estas tienen una diferencia de entre 4% - 6%.

Para el caso de *MLPClassifier* se observa (ver Tabla 2) que las métricas son semejantes en todas las categorías. Probablemente esto se debe a la generalización característica de las redes neuronales. Relacionado a la comparación con otros modelos se puede decir que el rendimiento fue adecuado y comparando con los modelos realizados en este proyecto se puede decir que el rendimiento fue medio, no fue el mejor pero tampoco el peor. Cabe mencionar que el entrenamiento fue tardado, probablemente porque no se lograba converger a un mínimo global para la pérdida. Esto fue un hecho ya que el algoritmo se detuvo en la época 30 al observar que las métricas de validación no mejoraba. Llama la atención que el modelo obtenga métricas tan semejantes en *precision* y *recall*. Este comportamiento nos indica que el denominador para estas funciones es semejante:

$$p = \frac{tp}{tp+fp}\ ;\ r = \frac{tp}{tp+fn}$$

$$p \simeq r \rightarrow fp \simeq fn$$

Esto puede ser debido a que la data estaba balanceada en la característica objetivo y también en algunas otras características como se observó en la exploración. Para el caso de XGBoost no hay mucho que discutir: es el modelo de los más populares entre la comunidad de *Data Science* y se esperaba en definitiva que tuviera mejores métricas que los otros modelos. Sin embargo, la diferencia de las métricas fue mínima aunque muy buena si se compara con métricas de otros competidores.

A la empresa Microsoft le interesa mucho el poder clasificar adecuadamente el malware pero también debe tomar en cuenta un pilar importante de la seguridad de información el cual es la disponibilidad. Es por ello que la métrica de *precision* toma singular importancia ya que se requiere que se clasifique adecuadamente una computadora con altas probabilidades de malware. Esto bajo la perspectiva que luego de clasificar a una computadora con la etiqueta de probabilidad alta de malware el usuario tenga que realizar tareas correctivas que puedan llegar a afectar su experiencia. Por otro lado, si el caso de uso va más orientado a no dejar pasar ninguna computadora con verdadero malware se podría considerar la métrica de *recall*. De esta manera se pueden llegar a etiquetar equipos que no tiene malware como positivos a malware pero se reduce la probabilidad de etiquetar sin malware a aquellos que sí tienen.

## Conclusiones

- *XGBoost* fue el mejor modelo para identificar máquinas infectadas por malware con una precisión de 0.6495, esto fue validado con validación cruzada *K-folds* con k=10 obteniendo puntajes muy cercanos al rendimiento real.
- La selección de características y preprocesado de la información fue en lo que más tiempo se invirtió, sin embargo, valió la pena ya que los modelos lograron comprender el conjunto de datos preprocesado con mayor facilidad logrando métricas favorables.
- Según referencias externas se logró crear modelos con métricas de clasificación adecuadas. La razón principal de no obtener métricas excelentes fue debido a la heterocedasticidad de la información además la no linealidad y la falta de información del contexto general en la que la computadora se encontraba restringe un poco que los modelos clasifiquen de forma adecuada.
- Se logró explorar los datos telemétricos de Windows. Con esta información se emplearon métodos de Ciencia de Datos con el fin de poder clasificar malware, esto nos brinda herramientas para poder defender la confidencialidad, la integridad y la disponibilidad de la información, siendo estos pilares importantes en la seguridad, ya que por lo general el malware como nombre genérico es utilizado para fines maliciosos no solo para el robo de información.
- Este proyecto brinda una parte a la solución de detección de malware, existen diversas herramientas vistas en clase que complementan esta solución con el fin de ofrecer la mejor experiencia al usuario y no dejando atrás la protección de la información.

## Recomendaciones

Se recomienda para futuras implementaciones poder derivar más información de las columnas al lado de un experto en telemetría de Windows con el fin de poder crear más datos e incrementar la probabilidad de que los modelos tengan otra perspectiva diferente de los equipos infectados.

## Anexos

El código utilizado para poder llevar a cabo el presente proyecto puede ser encontrado en el siguiente enlace: https://github.com/gusmendez99/sds-project-2.

## Referencias


[1] Gandhi, R (2018, 5 Mayo) Naive Bayes Classifier. Extraído de https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

[2] IBM. (2021, 17 agosto). El modelo de redes neuronales. El modelo de redes neuronales. Extraído de https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model

[3] Kaggle (2019). Microsoft Malware Prediction. Extraído de https://www.kaggle.com/competitions/microsoft-malware-prediction

[4] Kaspersky (2022). Machine Learning in Cybersecurity. Extraído de https://www.kaspersky.com/enterprise-security/wiki-section/products/machine-learning-in-cybersecurity

[5] Mahanta, J. (2018, 10 junio). Introduction to Neural Networks, Advantages and Applications. Extraído de https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207

[6] Microsoft (2017, 3 agosto). Windows Defender ATP machine learning: Detecting new and unusual breach activity. Extraído de https://www.microsoft.com/security/blog/2017/08/03/windows-defender-atp-machine-learning-detecting-new-and-unusual-breach-activity/

[7] Or-Meir, O., Nissim, N., Elovici, Y., y Rokach, L. (2019). Dynamic malware analysis in the modern era—A state of the art survey. *ACM Computing Surveys (CSUR), 52*(5), 1-48.

[8] Patel, V., Choe, S., y Halabi, T. (2020). Predicting future malware attacks on cloud systems using machine learning. *In 2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)* (pp. 151-156). IEEE.

[9] Stewart, M. P. R. (2021, 11 diciembre). The Limitations of Machine Learning - Towards Data Science. Medium. Extraído de https://towardsdatascience.com/the-limitations-of-machine-learning-a00e0c3040c6

[10] Windows Defender Research (2018, 13 diciembre). Microsoft AI competition explores the next evolution of predictive technologies in security. Extraído de https://argonsys.com/microsoft-cloud/library/microsoft-ai-competition-explores-the-next-evolution-of-predictive-technologies-in-security/

[11] XGBoost (2022). XGBoost Documentation. Extraído de https://xgboost.readthedocs.io/en/stable/