Malware Detection with Machine Learning

Adrián Chamorro Fernández

Resumen– Ante la necesidad de un método definitivo que pueda hacer frente a los hackers y su gran cantidad de ataques diarios, nace la idea de este proyecto, que se basa en utilizar la potencia de la inteligencia artificial como objetivo para conseguir mayor seguridad en los sistemas informáticos. En este informe se escogerán dos modelos de inteligencia artificial diferentes junto con un conjunto de datos para comparar sus resultados y se escogerá un modelo para aplicar mejoras mediante diferentes técnicas y así crear un modelo aún más robusto y efectivo contra los ficheros *malware*. Con los modelos escogidos y el desarrollado en este proyecto, se realizará una comparación con unas métricas previamente seleccionadas para evaluar el rendimiento de cada uno. En definitiva, se mostrará la capacidad de precisión que puede tener la inteligencia artificial junto con la ciberseguridad sin la interacción de ningún humano y las diferentes técnicas que conllevan a ello.

Palabras clave– Malware, Inteligencia Artificial, Ciberseguridad, Clasificación Binaria, Light Gradient Boosting Machine, Deep Neuronal Network, Métodos de Conjunto, Conjunto de datos

Abstract– Faced with the need to find a final method which deal with hackers and the huge daily number of attacks, it borns the idea of this project, which is based on use the power of artificial intelligence like a goal to get better security on computer systems. In this paper it will choose two different artificial intelligence models in conjunction with a dataset to compare their results and it will pick one model to apply improvements by different techniques and in this way create an even more robust and effective model against malware files. With the picked models and the one developed in this project, it will make a comparison with a previous chose metrics between each of them to asses performance results. In short, it will show precision capacity that it can have artificial intelligence with the help of cybersecurity without any human interaction and the different techniques which lead to it.

Keywords– Malware, Artificial Intelligence, Cybersecurity, Binary Classification, Light Gradient Boosting Machine, Deep Neuronal Network, Ensemble Methods, Datasets

1 Introducción

DESDE la última década, tanto el sector de la ciberseguridad cómo el de la inteligencia artificial han ido creciendo a pasos agigantados, convirtiéndose cada vez más en esenciales para realizar cualquier tipo de función dentro de nuestros sistemas informáticos, ya sea con una acción tan sencilla como utilizar el sistema de recomendaciones de una aplicación de contenido en streaming, o bien, cuando un antivirus tiene que detectar si los datos que se están almacenando en un servidor de ficheros están infectados o no.

Una de las tareas más significativas dentro de cualquier sistema informático que almacena datos, es la detección de ficheros maliciosos para asegurar la integridad, confidencialidad y disponibilidad del sistema en cuestión. Esta tarea actualmente se realiza mayoritariamente mediante la técnica de detección basada en firmas por los antivirus más comerciales [1]. Este procedimiento ha obtenido buenos resultados a lo largo de estos últimos años, pero actualmente existe una lucha constante contra hackers que, conociendo las técnicas de detección de *malware* tradicionales, intentan burlar constantemente estos controles e infectar cualquier tipo de sistema con los denominados *malware* de día cero [2]. Los *malware* de día cero son los que no han sido previamente detectados por ningún sistema y que, por tanto, ninguna base de datos de virus puede contenerlo para detectarlo a tiempo y eliminarlo.

Para intentar inclinar la balanza a favor en esta batalla ante los *malware* de día cero, se está intentando absorber nuevas tecnologías como la inteligencia artificial para resol-

• E-mail de contacto: adrian.chamorro@e-campus.uab.cat
• Mención realizada: Tecnologías de la Información
• Trabajo tutorizado por: Ana Oropesa Física (Departamento de Ingeniería de la Información y de las Comunicaciones)
• Curso 2019/20

ver estas detecciones a través de métodos heurísticos, con el propósito de poder adelantarse con detecciones precoces que protejan los sistemas con *malware* anteriormente no detectado a través de reconocimiento de patrones sobre estos ficheros imperceptibles para los humanos [3].

La inteligencia artificial abre nuevas vías para mejorar la ciberseguridad, pero al ser un campo todavía muy novedoso, no existe un modelo estándar que cubra todos los casos sin poder ser vulnerable al ataque de un hacker [4]. Sumado a que investigadores que trabajan en una solución que pueda acercarse a la perfección, en cuanto a seguridad se refiere, se encuentran a día de hoy, con impedimentos como la falta de datos con los que puedan entrenar sus modelos de detección de *malware* para conseguir los mejores resultados posibles y sobre todo realistas.

Ante las nuevas investigaciones que se realizan actualmente para conseguir mejorar estos modelos de detección de *malware*, se han unido en este trabajo los términos de ciberseguridad e inteligencia artificial, para poder realizar alguna investigación en este campo tan sumamente reciente. Se ha realizado un análisis de dos modelos encargados de la clasificación binaria para ficheros entre *malware* y cleanware, además de crear un nuevo modelo mejorado de uno de los ya vistos en la fase anterior de análisis. Este análisis e implementación del nuevo modelo, se ha realizado utilizando datos reales *open-source*, los cuales se han tenido que preparar y adaptar para su posterior uso para cada modelo. Al finalizar el entrenamiento de estos modelos para producir nuevos resultados sobre nuevos datos anteriormente no vistos, se han generado unas conclusiones con la ayuda esencial del uso de unas métricas específicas para algoritmos de clasificación binaria.

Este documento ha organizado toda la información de la siguiente manera, en la sección 2 se encuentra el estado del arte, dónde se determina el modo de cómo se ha tratado dicho tema, las tendencias actuales sobre la materia y los avances sobre su conocimiento. En la sección 3 se localizan los objetivos del proyecto, seguidamente en la sección 4 se explicarán la metodología y la planificación utilizada para este proyecto. Adentrándonos en el corazón del proyecto, se halla la sección 5 dónde se ha realizado el desarrollado de este trabajo y en el apartado 7 se presentarán los resultados obtenidos. Finaliza este informe con la sección 8 dónde se encuentran las conclusiones y la 9 con el trabajo futuro. También se incluyen una sección de agradecimientos y referencias bibliográficas.

2 ESTADO DEL ARTE

El campo de la inteligencia artificial aplicada a la ciberseguridad es actualmente un campo en crecimiento, ya que cabe decir que muchas de las estrategias que se han utilizado para abordar la detección de *malware*, han sido fallidas a causa de la aparición de nuevas respuestas para sobrepasar estas barreras anti-hackers [15]. Estas respuestas, aunque no todas [14], han sido desarrolladas mediante la misma tecnología: la inteligencia artificial[4].

Ante no tener una solución para la detección cien por cien independiente con inteligencia artificial sin poder ser vulnerable a ataques, los antivirus más grandes del mercado, como es el caso de Kaspersky[16], han decidido aplicar soluciones hibridas como es el caso de la detección mediante heurísticas.

Este trabajo se enfoca en el marco de los algoritmos encargados de la detección estática, es decir, sin ejecutar el fichero para saber si está infectado o no. Se escogerán dos algoritmos encargados de realizar este tipo de detecciones y se comparará su rendimiento. A continuación se trabajarán nuevas propuestas para mejorar el rendimiento de aquel algoritmo que peor resultados haya obtenido. De esta manera, se podrá observar en la evolución de este proyecto las soluciones con mejor rendimiento y cómo atacar un problema de estas dimensiones con un hardware limitado computacionalmente y con la combinación de estrategias para su mejora.

Conrectamente en este estudio, se ha basado en los algoritmos LightGBM, DNN y Malconv. LightGBM es un framework de aumento del gradiente que usa un algoritmo basado en la combinación de N árboles de decisiones que actuan cojuntamente para realizar predicciones de clasificación binaria.

La DNN se trata básicamente de una red neuronal, la cual recibe unos *inputs* y através de las capas que contienen un número determinado de neuronas, genera un *output*, que en este caso es binário.

MalConv es un algoritmo de detección de *malware* que utiliza directamente la secuencia de bytes del fichero y los introduce es su arquitectura que se compone de capas convulsionales, una red completamente conectada y una capa *softmax* para clasificar el resultado de la predicción.

Se tratará de mejorar uno de los modelos con peores resultados, con los denominados Métodos de Conjuntos. Los métodos de conjunto es una técnica que consta de utilizar varios modelos de predicción en conjunto como si fuera uno solo. Esta técnica divide los modelos que participan en esta unión en dos tipos: *weak learners* y *strong learner*. Estos dos tipos corresponden a los que trabajan para realizar la primera predicción y los que reciben dicha predicción para generar la salida final del Método de Conjunto.

Existen tres tipos de métodos de conjunto, según como se combinan los modelos base o *weak learners* [19] :

- *Bagging*: se compone generalmente de *weak learners* homogéneneos, es decir, que todos los modelos del tipo *weak learner* han de ser del mismo tipo. Trabajan en paralelo independientemente y combinan sus resultados para ofrecer la predicción a introducir en el *strong learner*.
- *Boosting*:se compone generalmente de *weak learners* homogéneos como en el tipo *bagging*. Trabajan en secuencialmente creando una cadena de predicciones que hace que cada modelo *learner* aprenda del anterior.
- *Stacking*: se compone generalmente de *weak learners* heterogéneos, es decir, que en este caso se puede utilizar diferentes tipos de modelos. Estos modelos trabajan en paralelo para producir las predicciones a introducir en el *strong learner*.

Para este proyecto se utilizará la ayuda de un modelo denominado *Random Forest*. *Random Forest* en si, se trata de un método de conjunto de tipo *bagging*. Es decir, siguiendo

la definición de los tipos *bagging*, *Random Forest* se compone de n *weak learners*, que en este algoritmo son árboles de decisión, que realizan sus predicciones y de un *strong learner* encargado de absorber estas predicciones para realizar una media, la cual será la predicción final [20].

Los resultados de este informe se valorarán a través de las siguientes métricas seleccionadas, ya que representan en su mayor capacidad el rendimiento de un modelo de clasificación binaria:

- Exactitud: Predicciones correctas sobre todas las predicciones realizadas.
- Clasificación errónea (CE): Predicciones incorrectas sobre todas las predicciones realizadas.
- Sensibilidad: Predicciones positivas acertadas sobre todos los datos positivos.
- Precisión: Predicciones acertadas sobre todas las predicciones positivas.

Para mostrar los resultados y compararlos de una forma más visual y clara, en este proyecto se utilizará la matriz de confusión. Esta matriz está compuesta de dos columnas y dos filas, es decir, únicamente hay cuatro valores.

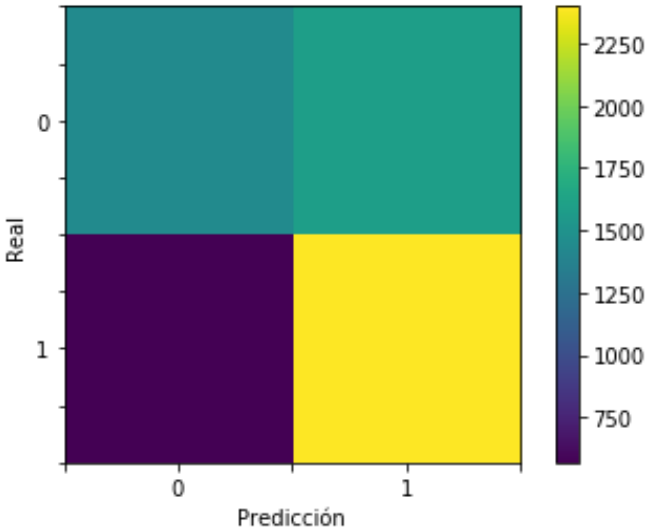

Fig. 1: Ejemplo matriz de confusión

Los valores son los que se pueden ver en la Fig. 1, los cuales están representados acorde con una escala de colores situada en la parte derecha. Tomando como ejemplo esta Fig. 1 se puede concretar lo siguiente:

- Veraderos Negativos (esquina superior izquierda): Existen 1586 ficheros sin *malware* dónde se ha predecido que no tienen *malware*.
- Falsos Positivos (esquina superior derecha): Existen 1620 ficheros que no tienen *malware*, pero se ha predecido que tienen malware.
- Falsos Negativos (esquina inferior izquierda): No existe ningún fichero que tenga *malware* y que el sistema ha predecido que no lo tiene.
- Verdaderos Positivos (esquina inferior derecha): Existen 2384 ficheros que tienen *malware* donde se ha predecido que tienen malware.

Como se puede observar con el ejemplo propuesto, en un sistema de clasificación binaria se intenta aumentar los valores Verdaderos Negativos y Verdaderos Positivos, así como minimizar los Falsos Positivos y los Falsos Negativos para obtener un mejor rendimiento del modelo realizando predicciones.

3 OBJETIVOS

El problema que se trata de estudiar en este documento, se puede abordar de múltiples formas. En este trabajo se ha realizado el planteamiento de los objetivos que se muestran a continuación:

- Analizar los diferentes modelos de detección de malware más utilizados según diferentes fuentes de información [5] [6] [7] [8] [9] antes de la fase de desarrollo, con el fin de elegir los dos mejores para su posterior entrenamiento.
- Analizar las métricas de eficiencia más representativas según las fuentes [5] [10] después de la elección de los modelos y antes del análisis de los datos, para la elección de las métricas que van a evaluar la eficiencia de los entrenamientos de los modelos escogidos.
- Analizar los diferentes conjuntos de datos publicados y accesibles más utilizados según los estudios realizados en las fuentes de información [5] [8] [9] después de la elección de las métricas y antes del entrenamiento de los modelos, con la finalidad de escoger un conjunto de datos y poder entrenar los modelos con estos archivos reales y así mejorar su rendimiento ante su puesta en producción.
- Generar el análisis de los resultados dadas las métricas mediante librerías de programación en Python, dónde se pueda ver de una forma clara y concisa la eficiencia de los dos modelos después del entrenamiento de estos y antes de la selección de un modelo con posibles mejoras, para poder comparar el rendimiento de los dos modelos sobre las métricas escogidas.
- Seleccionar un modelo con posibles mejoras basándose en el análisis generado en el punto anterior observando su inferioridad comparando con el modelo contrario después de la generación de análisis de resultados y antes de la investigación sobre las mejoras a realizar, con el propósito de mejorar los resultados sobre las métricas.
- Investigar sobre posibles modificaciones al modelo elegido anteriormente inspiradas en estudios previos sobre la detección de malware después de la selección de dicho modelo y antes del desarrollo de estas mejoras, para plantear posibles mejoras sobre el modelo.
- Implementar modificaciones al modelo y entrenarlo con las modificaciones realizadas utilizando un entorno con procesamiento GPU, midiendo su eficacia con las métricas seleccionadas previamente después de investigar sobre posibles mejores en el modelo y antes del análisis de los datos entre los tres modelos, para mejorar su desempeño respecto al modelo sin modificaciones con la detección de malware.

- Analizar los resultados entre los dos modelos iniciales y el modelo escogido con la mejora aplicada junto con las posteriores conclusiones, mediante librerías de programación en Python, dónde se pueda ver de una forma clara y concisa la eficiencia y/o diferencia de los tres modelos después de realizar las modificaciones en el modelo escogido, para poder comparar el rendimiento de los diferentes algoritmos sobre la detección de malware.

4 METODOLOGÍA Y PLANIFICACIÓN

Para la organización y distribución de trabajo, así como la priorización y la visualización clara de los objetivos, se utilizará un diagrama de Gantt. El diagrama de Gantt es una de las herramientas más utilizadas en la gestión de proyectos [11]. El diagrama permite visualizar las actividades y el camino crítico de nuestro proyecto de una forma gráfica y simple. Utilizando esta herramienta se podrá abordar problemas como el camino crítico, el cual permite estimar cuánto tiempo se puede demorar una actividad para lograr el objetivo de llegar al *deadline* habiendo completado todas las actividades previstas.

Los períodos cronológicos para completar estas actividades se han dividido en semanas y cada semana en subperíodos de cuatro días, ya que se estima que se trabajará en el proyecto una media de cuatro días por semana. Haciendo esta distribución y dejando un tiempo en cada tarea para imprevistos, se completarán las tareas utilizando un modelo en cascada [24].

5 DESARROLLO

En esta sección se muestran las diferentes acciones paso a paso que se han realizado para poder conseguir los objetivos fijados previamente. Dentro de este apartado, se encuentran dos fases.

5.1. Fase I

Esta fase consta de cuatro apartados, comenzando por la investigación y análisis del problema, seguido de la elección y tratamiento del conjunto de datos, para poder llegar hasta el entrenamiento de modelos y finalmente generar los resultados obtenidos.

5.1.1. Investigación y análisis

Para adentrarse en el desarrollo de este proyecto, previamente se ha realizado una investigación sobre las diferentes formas de detección posibles sobre un fichero. Sobre esta línea se ha constatado que existen dos métodos principales de detección de *malware*: basadas en firmas o basadas en heurísticas.

En el método donde se pone el foco es en el método basado en heurísticas, ya que es el que está acabando de asentarse y que todavía queda mucha mejora para aprovechar todo el potencial que puede ofrecer. Dentro de este método, se encuentra otra subdivisión, esta vez de técnicas de detección de *malware*. Nos encontramos con las técnicas estáticas y las técnicas dinámicas. Las técnicas estáticas, a diferencia de las dinámicas, son aquellas que sin necesidad de ejecutar el fichero analizado, es capaz de examinar la estructura de las cabeceras y del contenido para asignarle una predicción. Esta última técnica es la que se utiliza para este proyecto basándose en el riesgo que supone utilizar una detección dinámica la cual se ha de ejecutar un código malicioso de forma aislada sin afectar al resto del sistema.

Una vez escogido el método y la técnica a utilizar, se ha procedido a la investigación sobre los diferentes tipos de algoritmos/modelos que ya existen y están en desarrollo. Uno de los algoritmos más utilizados y con mejores resultado es el denominado como MalConv [12]. Esta fue la primera elección al ver el alto rendimiento que tenía, pero en busca de un conjunto de datos que pudiera satisfacer las necesidades de esta arquitectura se llegó a una conclusión de que los datos públicos que contengan ficheros ya clasificados no abundan. Es por eso, que se procedió a la creación de un conjunto de datos propio utilizando un dataset público de Microsoft cómo ficheros infectados y con la extracción de los ficheros *Portable Executables* de una máquina virtual de Windows 10 se obtuvieron los ficheros benignos. Dada la complejidad de construir correctamente un conjunto de datos propio desde cero, que permita al algoritmo generalizar respecto a la detección de nuevos datos anteriormente no vistos y a causa de la falta de recursos, se ha procedido a seguir investigando sobre otros algoritmos, abandonando de esta manera la idea de trabajar con MalConv.

Entonces fue cuando el modelo Light Gradient Boosting Machine (LGBM) en una de las investigaciones analizadas, superó a MalConv [5]. LGBM es un framework de aumento del gradiente que usa un algoritmo basado en árboles de decisiones. Se escogió LGBM para trabajar en este proyecto dada su rapidez y efectividad [13]. Por otro lado, en este proyecto se utiliza Deep Neuronal Network (DNN), cómo modelo principal para analizar los resultados de una versión inicial y dónde más tarde se centrarán los esfuerzos en el desarrollo de mejoras que puedan conllevar a detecciones más potentes.

5.1.2. Conjunto de datos

Para acotar y centrar los esfuerzos en un tipo de datos existiendo el gran abanico de formatos en los que nos podemos encontrar camuflado un código malicioso, se ha escogido los ficheros *Portable Executables* (PE) de unos de los conjunto de datos *open-source* más completos utilizados en una gran cantidad de investigaciones en la detección de *malware*.

El conjunto de datos, tal como se muestra en la Fig. 2, consta de un total de 1M de datos, los cuales se divididen en 800.000 para la fase de entrenamiento y 200.000 para la fase de test. Si clasificamos por tipo, tenemos tanto *malware* y *cleanware* con 300.000 datos cada uno para la fase de entrenamiento y 100.000 cada uno para la fase de test.

Una vez escogido el conjunto de datos a utilizar, se han tratado para poder utilizarlos en los diferentes modelos que se han ido implementando y entrenando. Esto es debido a que al ser unos datos tan pesados, la fuente de estos datos ofrece una serie de ficheros *json* para no centralizar todos los datos en un solo fichero, que se deben de centralizar en un solo objeto memmap en el caso del modelo LGBM y en un tensor en el caso de la DNN para poder hacer uso de ellos.

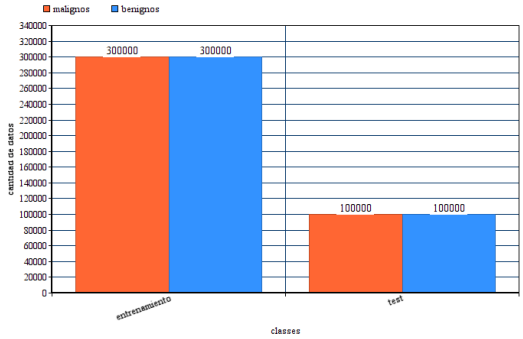

Fig. 2: Distribución de los datos

5.1.3. Hardware

Una vez escogidos los modelos y el conjunto de datos con los que trabajar, se ha de proceder con el entrenamiento. Este período implica una parte *hardware*, la cual se ha tenido que tener muy en cuenta para poder realizar dichos entrenamientos y la preparación que estos suponen.

El *hardware* con las que se han realizado las primeras pruebas fue el compuesto por una tarjeta gráfica NVIDIA GTX-1070 Max-Q de 8Gb, un procesador Intel(R) Core(TM) i7-8750H y una memoria RAM de 16Gb. Dividiendo los datos en lotes para no saturar la RAM, se han obtenido unos resultados negativos, ya que en todo momento el modelo intenta reservar más memoria que la que tiene a su alcance. En esta situación, se han planteado dos opciones: reducir el tamaño de los datos, lo cual conllevaría tiempo analizando los datos más significativos para obtener los resultados más generalizados o bien, recurrir a uno de los recursos más utilizados como es el *cloud*.

En este proyecto se ha optado por la opción *cloud*, en concreto, Google Cloud [23]. Dentro de esta plataforma se ha configurado una instancia de 24 CPUs y 90Gb de RAM.

5.1.4. Entrenamiento de modelos

Con el hardware ya definido, se ha procedido a realizar los entrenamientos en la instancia creada del *cloud*, tanto con el modelo LGBM como con la DNN.

En el caso del model LGBM, se ha realizado la técnica denominada optimización de parámetros, ya que al ser un algoritmo donde a diferencia de una regresión lineal, contiene una gran posibilidad de parámetros y existe una gigantesca lista de parámetros con la que se puede configurar nuestro algoritmo. La función que realiza es que dados unos parámetros y una lista de posibles valores para esos parámetros, el proceso de optimización automáticamente entrenará el modelo con todas las posibles combinaciones y seleccionará aquella combinación que haya obtenido los mejores resultados según la métrica de curva de característica operativa del receptor (ROC) calculada en cada combinación.

Al trabajar con redes neuronales como la DNN que se utiliza en este proyecto, entre las capas se genera información y esta información, como es de suponer, conlleva capacidad de almacenamiento. Incluso al tener 90Gb de RAM con la instancia creada en Google Cloud, se han tenido que intentar realizar entrenamientos con más de cinco prototipos con diferentes capas para poder llegar a una arquitectura que es capaz de computacionalmente conseguir el objetivo deseado, el entrenamiento de los datos.

5.1.5. Generación de resultados

Acabada la etapa de entrenamiento, se extraerán los resultados mediante las métricas de exactitud, clasificación errónea (CE), sensibilidad y precisión.

Estas métricas son las más utilizadas en algoritmos de clasificación binaria. Todas estas métricas se basan en cuatro métricas base que se pueden observar en la Tabla 1, la cual se denomina matriz de confusión.

	Predicción negativa	**Predicción positiva**
Real negativo	Verdaderos Negativos	Falsos Positivos
Real positivo	Falsos Negativos	Verdaderos Positivos

TABLA 1: MÉTRICAS BASE

Uno de los objetivos más importantes del algoritmo LGBM y DNN se basa en minimizar la tasa de Falsos Negativos (FN) y Falsos Positivos (FP). Tal como se muestra en la Fig. 3 para el modelo LGBM se han conseguido unos resultados notables con una exactitud del 0.9473 mientras que en DNN únicamente hemos obtenido un 0.5007. Únicamente por el resultado de la precisión con la DNN, podemos concluir rápidamente que se está produciendo un ajuste bajo, es decir, que no está generalizando bien.

Se puede afinar más este análisis, observando el valor de precisión de la DNN con un 0.9979 podemos decir que la mayoría de las predicciones positivas que ha realizado han sido acertadas en un 0.9979, pero en cambio la sensibilidad nos dice que de todos los casos positivos que tenemos en nuestro conjunto de datos, únicamente se ha acertado el 0.0035.

De una forma más sencilla podemos observar como en Fig. 4 se observa la matriz de confusión para LGBM y en Fig. 5 para DNN.

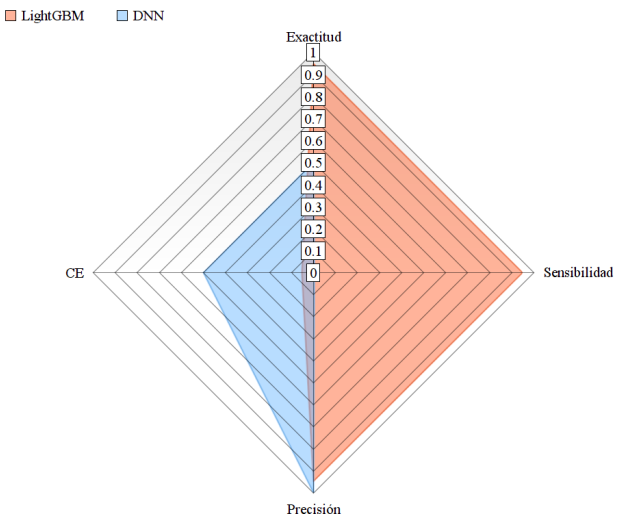

Fig. 3: Resultados entrenamiento inicial

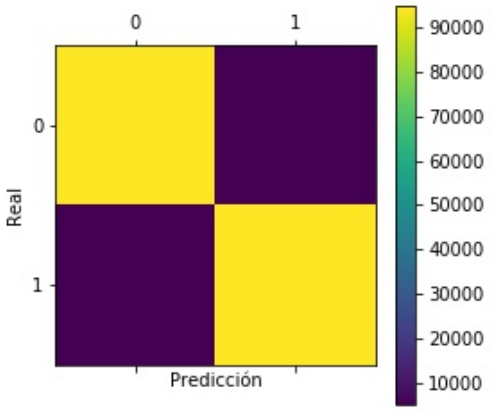

Fig. 4: Matriz de confusión para LGBM

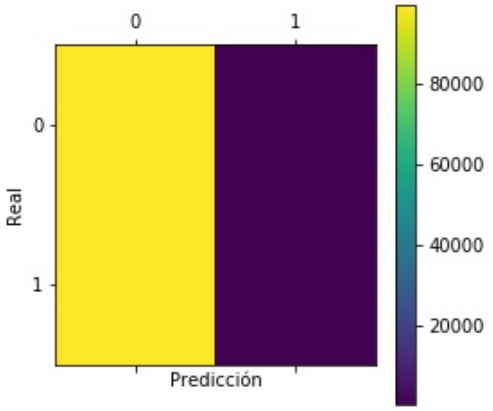

Fig. 5: Matriz de confusión para DNN

6 Fase II

6.1. Consecuencias COVID-19

Ante la aparición de la pandemia a nivel mundial provocada por el COVID-19, se han tenido que redefinir algunos de los objetivos inicialmente planteados para este proyecto.

El factor que hace que esta situación de pandemia afecte al proyecto, es debido a la limitación hardware existente para la realización de los cálculos necesarios y la posterior obtención de resultados concluyentes que permitan analizar estos modelos de detección de *malware* planteados en este documento.

Como se puede observar en el apartado 5.1.3 Hardware, la elección del hardware fue decisiva antes de realizar las primeras implementaciones. Es por eso, que en aquel momento se escogió la nube, en concreto, Google Cloud con una instancia virtual de 24 CPUs y 90 Gb de RAM. Con este hardware se pudo realizar sin problemas la Fase I de desarrollo de este proyecto utilizando 1M de datos para la realización de los cálculos pertinentes.

El 7 de Mayo de 2020, al probar de inicializar la instancia, Google Cloud informó de que la cuota límite de CPUs se redució únicamente a 8 CPUs. En ese momento, y ante la posibilidad de uso de esa instancia, se plantean varias alternativas para poder continuar con el proyecto y llegar al *deadline* con el mínimo de afectaciones sobre los objetivos iniciales definidos.

Entre todas las situaciones planteadas, se decanta sobre la opción de trabajar con la máquina local, también expuesta en la sección 5.1.3 Hardware. Obteniendo de esta manera, una potencia de 12 CPUs y 16 de RAM con una tarjeta gráfica (GPU) NVIDIA GTX-1070 Max-Q de 8 Gb dedicados. Obviamente pasar de 24 CPUs a 12 CPUs, hace que exista un cuello de botella bastante grande para poder llegar al *deadline* marcado. Ante la falta de recursos, se plantea una inversión de tiempo en rediseñar los modelos expuestos en la Fase I con el claro objetivo de utilizar el recurso más preciado de la máquina local, como es la GPU. Invirtiendo un buen bloque de tiempo en la utilización, cuando ha sido posible, de la GPU y con una reducción de datos se ha podido llevar a cabo la actual Fase II.

6.2. Entrenamiento de modelos de la Fase I

Dada la explicación del apartado anterior sobre las consecuencias del COVID-19 en este proyecto, se ha optado por volver a diseñar y entrenar los modelos expuestos en la Fase I: LightGBM y DNN.

6.2.1. Conjunto de datos

Para volver a rediseñar los modelos se ha recurrido a diferencia de las implementaciones de Fase I para la DNN, al uso de hiperparametros con su debía optimización. Este proceso de optimización de parámetros para cada uno de los modelos respectivamente, como ya se ha comentado, es computacionalmente muy costos. Por esta razón, se ha procedido a realizar una reducción del conjunto de datos.

Para realizar la reducción en el conjunto de datos (1M inicialmente), se ha recurrido al uso de la librería pandas [21] para la manipulación de estos. Concretamente, se ha procedido a seleccionar aquellos ejemplos cuya información sea más completa, es decir, aquellos que contienen menos referencias nulas en sus columnas (características). Además, se ha tenido que controlar que en los nuevos datos existiera una distribución lo más igualitaria sobre los dos tipos de clases: archivos benignos y archivos malignos.

El resultado de este procedimiento no fue directo, ya que en diferentes pruebas iniciales al producir esta reducción de datos, se encontraron igualmente deficiencias en el Hardware para poder realizar los cómputos adientes. Pero tras varias pruebas, se consiguió un buen subconjunto de datos de 36.000 ejemplos para el entrenamiento y 7.200 para la parte de test. La distribución en detalle del subconjunto de datos generados se puede observar en la Fig. 6.

6.2.2. Entrenamiento de los modelos

El siguiente paso ha sido, centrado principalmente en la GPU, el diseño de los algoritmos LightGBM y DNN. En el caso de LightGBM, no se ha tenido que realizar modificaciones en el modelo como tal para poder reentrenarlo con el nuevo subconjunto de datos. Aunque si se llevó a cabo de nuevo, una optimización de parámetros con 12 CPUs que

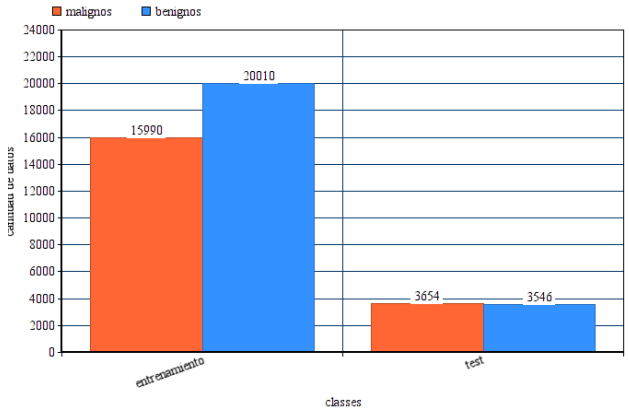

Fig. 6: Distribución del subconjunto de datos

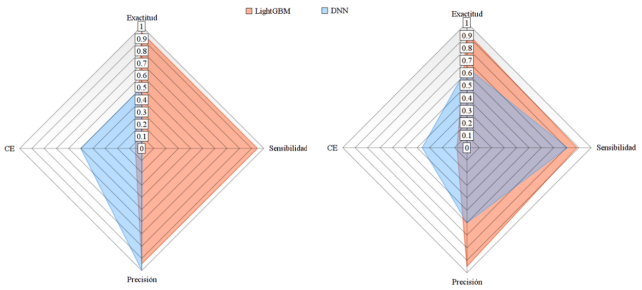

Fig. 7: Comaración de los modelos LightGBM y DNN en la Fase I (izquierda) y en la Fase II (derecha)

fue acompañada de un largo tiempo de computación al no tener posibilidad de optimización con GPU.

En cambio, en el modelo DNN si que se han realizado modificaciones importantes. Estas modificaciones, como ya se ha comentado, se han centrado en la posibilidad del uso de la tarjeta gráfica disponible en la máquina local, es por eso que se cambió el *framework* con el cual se estaba desarrollando, ya que el anterior utilizado en la Fase I, no permitía el uso de GPUs. Al igual que con LightGBM, se realizó una optimización de parámetros para buscar la mejor combinación de parámetros, incluyendo las capas escondidas, la tasa de aprendizaje, el número de neuronas entre capas y la tasa de abandono de los datos.

6.2.3. Generación de resultados

Los resultados de esta sección, utiliza las mismas métricas que la sección 5.1.5 Generación de resultados, ya que se trata de algoritmos de clasificación binaria y muestran muy notoriamente los resultados obtenidos.

Después de la adaptación de estos dos modelos ante la falta de recursos de cómputo, se han obtenido resultados, comparado con los desarrollados en la Fase I, muy parecidos en el caso de LightGBM, pero diferentes en el caso de DNN.

Comparando con la Fase I, como se puede observar en la Fig. 7, LightGBM ha obtenido resultados aproximadamente iguales en la métrica de exactitud, precisión y CE. En cambio, para la métrica de sensibilidad se denota una ligera disminución entorno a un 6 %, debido a la utilización de un conjunto de datos diferente.

Se observa que en el caso de DNN aumentan los valores de las métricas de exactitud y sensibilidad, y disminuye la precisión y el CE. Estos cambios tan notables en este modelo, se atribuyen a una variación drástica de los datos. Los datos fueron analizados y seleccionados para la Fase II a causa de las consecuencias del COVID-19, eliminando redundancias y ejemplos incompletos que como resultado en una red neuronal hace que pueda aprender con menos ruido en los datos y por tanto, con una mejor calidad que por supuesto, repercute positivamente en el proceso de aprendizaje del modelo.

Ademas de la comparación de los gráficos radar de la Fig. 7, donde se pueden contrastar todos los datos aportados en la explicación anterior, también se puede contemplar en la Fig. 8 y Fig. 9 las matrices de confusión correspondientes a los modelos LGBM y DNN respectivamente. Para poder procesar los resultados en el caso del modelo DNN se han utilizado 5590 ficheros y 7200 en el caso de LGBM. Esta diferencia es debida a la complejidad de cómputo en el caso del DNN. Aún así, en estas matrices comparándolas con las de la Fase I, se puede observar un peor resultado en el caso de los valores verdaderos positivos de LGBM, es decir, que el modelo no ha detectado tantos *malware* como en el caso de la Fase I. Y en el caso de DNN se puede contemplar una buena mejora tanto en el caso de los falsos negativos, como en los verdaderos negativos, ya que en la anterior versión se vio como se producía un ajuste bajo y no predecía del todo correcto los valores reales positivos del conjunto de datos.

Si se comparan entre ellos mediante las matrices de confusión se ve una clara mejora en términos generales de LightGBM sobre DNN, ya que no se ha de olvidar que LightGBM es un modelo que contiene pequeños predictores dentro en forma de árboles de decisión que hace que sea un algoritmo muy eficaz en este tipo de detecciones sobre una DNN que aprende a través de información procesada por neuronas con diferentes pesos.

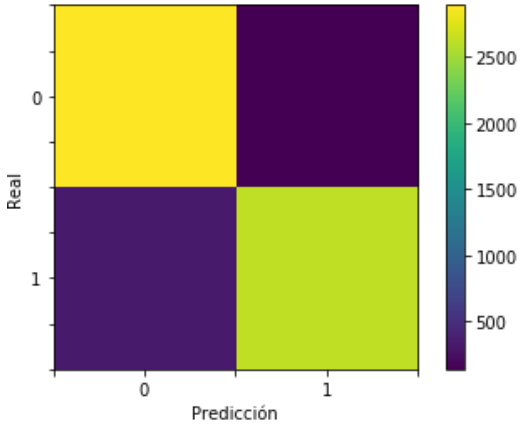

Fig. 8: Matriz de confusión para LGBM

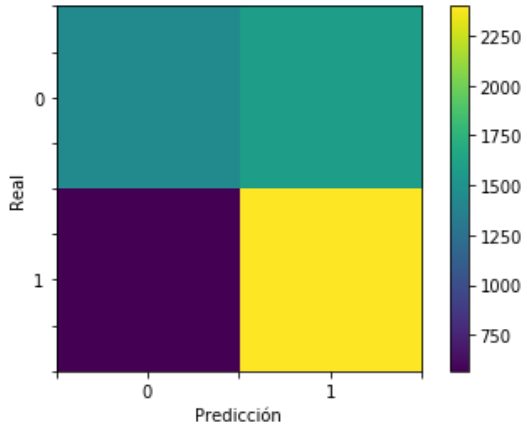

Fig. 9: Matriz de confusión para DNN

6.3. Mejora del algoritmo objetivo

Después de analizar los resultados expuestos en la Fase I de los dos modelos junto con, se procede a realizar una mejora en la red neuronal (DNN). Para mejorar el rendimiento del modelo seleccionado y viendo que en los resultados se detecta un claro sobre ajuste, se han aplicado diferentes técnicas que se muestran en los siguiente apartados.

6.3.1. Selección de características

Una de las acciones realizadas para la selección de características, se realizó ya en la sección 5.1.2 Conjunto de datos. Esta acción consistía en eliminar aquellos ejemplos que no portaban ninguna clasificación previa en ella, es decir, los archivos que inicialmente no estaban etiquetados como malignos o benignos.

Otra de las técnicas aplicadas, es una técnica fundamentada en observar que característica tiene más importancia que otra según un modelo determinado, en este caso, se ha utilizado *Random Forest*, ya que es uno de los modelos que se utilizarán en la sección 6.3.3 Métodos de conjunto. El resultado de aplicar esta técnica la podemos visualizar en la Fig. 10.

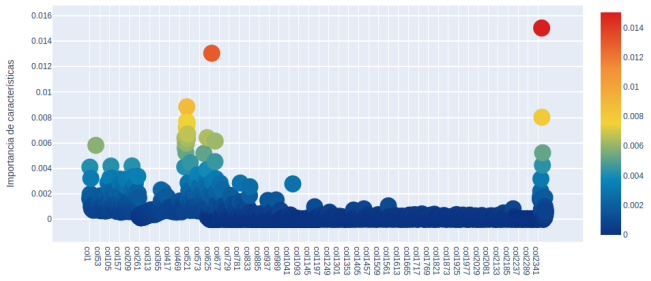

Fig. 10: Importancía de características para Random Forest

Por último, se realizó una técnica basada en eliminar todas aquellas características que tienen una varianza aproximada a 0. Esta técnica también incluye la exclusión de aquellas características que en todos los ejemplos propuestos tienen el mismo valor y, por tanto, no aportan nada nuevo a los datos.

Comentar que en este apartado se puede profundizar más aplicando diferentes técnicas y comparando los diferentes conjunto de datos que se pueden ir generando para observar las mejoras, o no, que estas técnicas nos puedes ofrecer. No se han realizado en este proyecto a causa de la falta de recursos en tiempo. Es por eso, que este apartado se añade al apartado de trabajo futuro.

6.3.2. Parada temprana

Ya se ha comentado anteriormente que se ha utilizado una optimización de parámetros para el mejor rendimiento del modelo sin que llegue a producirse un sobre ajuste en los datos o un ajuste bajo. En todo entrenamiento de un modelo, aparece un parámetro a utilizar esencial. Este parámetro se denomina en inglés *epochs*, es decir, el número de veces que se pasarán todos los datos al modelo para su aprendizaje, y para que la asignación de este valor aleatoriamente no sea un cuello de botella que produzca el mal funcionamiento del modelo al producir un sobre ajuste en los datos o un ajuste bajo, se aplica la denominada técnica parada temprana.

Los *epochs* se podrían ver como iteraciones del proceso de entrenamiento. Dónde en cada iteración se le pasarían todos los datos que le hayamos indicado. Por tanto, esta técnica trata de que cuando un modelo deja de mejorar en cada iteración, el entrenamiento se para. Esto permite quedarnos con el modelo que mayor resultado ha obtenido respecto a una métrica en concreto que se elija, por ejemplo, la precisión [22].

6.3.3. Métodos de conjunto

La última técnica planteada, la cual se denominará para el resto de este informe como AN por las última sílaba del autor de este proyecto, para la mejora de la red neuronal se trata de juntar dos modelos, para que las predicciones del primer modelo sirvan como entrada del siguiente, es decir, realizar una combinación lineal entre ambos [17]. Esta combinación permite obtener predicciones sobre los datos de mayor precisión al no contar con un solo modelo para realizar las predicciones, sino con una combinación de diferentes tipos que ayudarán a crear un modelo mucho más robusto. Esta técnica conocida en inglés como *Ensemble Methods*, generalmente se compone de la combinación de dos grupos.

Por una parte, tenemos en el primer grupo los denominados *weak learners* o modelos base. Los *weak learners* o modelos base se compone de un conjunto de modelos, los cuales realizan cada uno sus propias predicciones sobre los datos originales para tener diferentes puntos de vista. Por otra parte tenemos el *strong learner*, que conforma la última etapa del *Ensemble Method*. Esta etapa se constituye tan solo de un modelo que realizará las predicciones finales para clasificar el archivo como benigno o maligno.

Existen tres métodos de conjunto según como se combinan los modelos base: *bagging*, *boosting* y *stacking*. En este caso, se expondrá únicamente la definición de *stacking*, ya que es el método de conjunto utilizado.

La composición del método de conjunto implementado, por razones computacionales, está compuesto de un único weak learner que en este proceso es un modelo *Random Forest*. El *strong learner* será la red neuronal de la sección I. Esta implementación aún tener un único modelo base, al ser un *Random Forest* y como el objetivo se basa en mejorar los resultados de la red neuronal, desde el punto de vista teórico es una buena elección, al tratarse de un algoritmo

que ya por definición propia es un método de conjunto de tipo *bagging* tal y como se ha mencionado en el apartado del Estado del Arte, lo cual beneficia el hecho de tener más ”puntos de vista”.

Por último, en la última etapa, una vez realizadas las predicciones de nuestro *weak learner*, se introducirá a la red neuronal estas predicciones para aprender del error de ellas. Antes de realizar directamente el entrenamiento con el modelo ya generado en la Fase I, primero se ha realizado una optimización de parámetros, ya que se están cambiando los datos con los que inicialmente íbamos a entrenar a nuestra red neuronal. Esta optimización de parámetros permite que se obtenga una propuesta de parámetros para introducir a nuestra red neuronal y mejorar el rendimiento para estos nuevos datos provenientes de *Random Forest*.

De hecho, cabe decir que el modelo LightGBM utilizado también en la Fase I utiliza esta técnica de una forma transparente, de una forma muy parecida a *Random Forest*, sólo que en el caso de LightGBM se trata te un tipo *boosting*. Es decir, en vez de realizar los cálculos de los árboles de decisión en paralelo, los realiza secuencialmente [18].

Añadir que esta técnica ha sido desarrollada utilizando 12 CPUs en el primer modelo y 1 GPU en el segundo.

7 Generación de resultados

En esta sección se presentan la comparación de los resultados obtenidos durante todo este proyecto, enfocado en los modelos LightGBM, DNN y AN, desarrollados en la Fase II.

Podemos observar en la Fig. 11, la cual engloba los resultados de los tres modelos comparados con el modelo desarrollado AN. El modelo AN ha obtenido unos resultados más que satisfactorios desde la base del modelo DNN inicialmente seleccionado para la aplicación de mejoras. Esta mejora esta cuantificada aproximadamente en un 50 % sobre la DNN y en un 15 % sobre LightGBM. Se puede justificar este rendimiento superior de AN sobre DNN y LightGBM, debido a que nace de una combinación de ambos. Es decir, AN nace de la idea de utilizar las mejores propiedades de una red neuronal como DNN por la complejidad y efectividad de aprendizaje que tiene, junto con las propiedades de uno de los mejores modelos de clasificación binaria como *Random Forest*. Esto hace que se cree un modelo que debido a la utilización de los métodos de conjunto, aprenda no sólo de los errores de la misma DNN , sino que también de los errores que comete el modelo *Random Forest*.

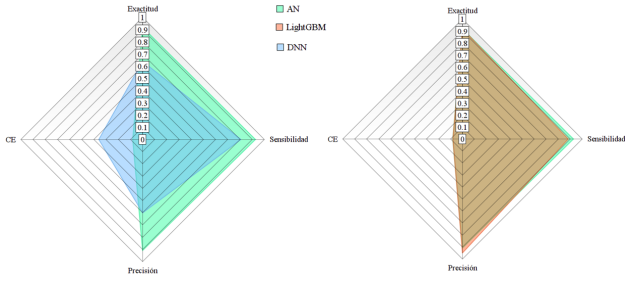

Fig. 11: Comparación entre los modelos AN y DNN (izquierda) y los modelos AN y LightGBM (derecha).

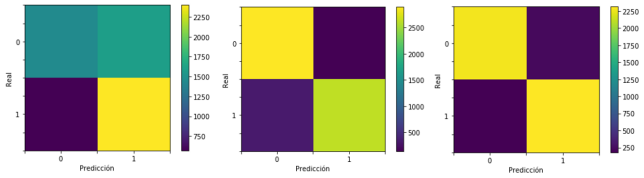

Fig. 12: Matriz de confusión para el modelo DNN (izquierda), LightGBM (centro) y AN (derecha)

Estas mejoras también se pueden apreciar si se contempla la Fig. 12 con la comparación de los resultados en forma de matriz de confusión, donde se puede apreciar los resultados muy positivos del modelo AN, ya que los valores falsos positivos y falsos negativos se minimizan consiguiendo una tasa de acierto del 92 %. Fácilmente se pude observar esta conclusión, definiendo como una matriz perfecta, una matriz con la esquina superior izquierda e inferior derecha completamente amarilla, que corresponden a los valores verdaderos negativos y verdaderos positivos respectivamente. El resto, es decir, las esquinas superior derecha e inferior izquierda completamente de color lila, al tratarse de los valores falsos positivos y falsos negativos respectivamente.

8 Conclusiones

En el momento de plantear que algoritmos utilizar en la investigación inicial, y al no encontrar ningún conjunto de datos que pudiera satisfacer las necesidades de una arquitectura tan poderosa como MalConv, fue cuando se extrajo una de las conclusiones más valiosas de este proyecto: las grandes empresas privatizan sus datos y no contribuyen en esta continua investigación en busca de un modelo definitivo en la detección de *malware*. Si bien es cierto, no es tan fácil como hacer público estos datos, sino que requiere un proceso de extracción de la parte más privada de los sistemas que contienen estos datos.

Un detalle muy importante que no se puede dejar pasar en cuanto a los conjuntos de datos, es el hecho de realizar un análisis exhaustivo antes de todo de los datos, para conseguir unos datos que representen con mayor claridad los diferentes tipos de clases y características que hacen que un modelo pueda realizar con seguridad sus predicciones. Esta conclusión viene derivada de la mejora inesperada en el desarrollo de la DNN de la Fase I con los nuevos datos generados, ya que claramente los resultados obtenidos fueron del todo positivos cuando únicamente se realizó una limpieza exhaustiva de los datos para poder realizar la reducción de estos. Si bien es cierto que estos análisis son un 60 % del tiempo dedicado en el proyecto, realizarlos ha sido crucial para la obtención de unos resultados tan positivos.

En el apartado de hardware, una de las elecciones relevantes para el desarrollo de este proyecto ha sido la de escoger a Google Cloud como hardware principal. Esta elección surgió como solución al imprevisto de falta de potencia computacional que se describe en el apartado 5.1.3. Como se puede observar, utilizar *cloud* para el hardware era la opción más rápida ante utilizar una máquina con ejecución local más limitada computacionalmente.

Si bien es cierto que el *cloud*, no fue la solución definitiva que se planteó inicialmente a causa de los argumentos explicados en la sección 6.1 Consecuencias COVID-19.

Asi que al final, si se tuvo que realizar el proyecto con la máquina local anteriormente mencionada con limitaciones computacionales. Aun así, el hecho de tener la obligación de realizar un rápido desarrollo para llegar a tiempo a la finalización de este proyecto, se tuvo que rediseñar todo el sistema de cómputo para poder aprovechar la GPU que esta máquina disponía. Y de esta acción, se puede extraer la conclusión que para una investigación de gran envergadura como esta, es necesario un cómputo alto que seguramente te podrá proporcionar una GPU o bien un conjunto de estas, debido a su gran rapidez y precisión a la hora de procesar cálculos.

Respecto a los resultados obtenidos en este proyecto son resultados en un entorno controlado y con una limitación muy importante en la cantidad de datos utilizados, a causa de otra limitación como es la del cómputo. Esto nos lleva a obviamente a realizar conclusiones dentro de este entorno. Quiere decir que en otro entorno con más variabilidad de parámetros, estos resultados podrían sufrir alteraciones.

Con los resultados planteados en este informe, podemos concluir que el mejor modelo para la detección de *malware* ha sido el desarrollado en la última fase, es decir, el modelo AN, ya que obtiene una de las tasas más altas de detección positiva de archivos malignos, así como de benignos y que por tanto hace que las tasas de errores sean mínimas. Como ya se ha comentado, esto es consecuencia directa de la aplicación entre otras técnicas, de los métodos de conjunto, que hace que un modelo muy bueno trabajando con problemas de clasificación binaria pueda alimentar con sus predicciones a otro modelo encargado de aprender de ellas y cuyo resultado es el que se ha podido observar en este proyecto con un gran rendimiento.

Con estos resultados positivos a favor del algoritmo AN, se puede afirmar que se han completado todos los objetivos planteados en el inicio del proyecto e incluso algunos que no fueron incluidos, como la superación en rendimiento al modelo LightGBM que va en cabeza en algunas investigaciones importantes en esta continua investigación sobre la combinación de la ciberseguridad con la inteligencia artificial.

Por último, se puede afirmar con completa seguridad que el uso de los *buffers* de tiempo definidos en la planificación inicial, han hecho que los imprevistos tan importantes como la afectación del COVID-19 en este proyecto, se hayan podido superar sin tener que haber salido de plazo en ninguna de las entregas realizadas.

9 Trabajo futuro

Como trabajo futuro, se puede incluir varios puntos, ya que la mayoría de estos no han sido aplicados en este proyecto debido a las limitaciones hardware mayoritariamente:

- Aplicar la técnica de reducción de selección de características mediante una matriz de correlación que permita ver que características tienen una dependencia más fuerte.

- Aplicar la técnica de reducción de la dimensionalidad de las características con el algoritmo *Principal Component Analysis* (PCA), para ver cuanto se puede mejorar a la hora de aligerar carga en los datos y centrarse en aprender características más selecta de los datos disponibles.

- Utilizar diferentes configuraciones en los *Ensemble Methods*, ya que esta técnica podría producir con más *weak-learners* una mejora notable en los resultados.

Agradecimientos

Primero de todo, agradezco el gran trabajo de mi tutora Ana Oropesa Física por su dedicación desde el minuto cero con este proyecto, por el gran servicio que me ha ofrecido durante todo el transcurso del trabajo y por haberme ofrecido su punto de vista en los momentos más complicados ayudándome a conseguir con éxito todos los objetivos planteados desde un inicio.

En segundo lugar, tengo que hacer mención a todos mis familiares por apoyarme y escucharme en mis momentos más complicados, especialmente a mi pareja Andrea por las facilidades que me ha proporcionado en el día a día. Incluyo también a Antonio Bayonas e Iván Lago por las recomendaciones sobre el uso del hardware en la nube.

Y por último, agradecer a todas aquellas investigaciones que están referenciadas en este informe y también aquellas que no lo están, que luchan día a día con pasión por lograr convertir la inteligencia artificial en la mejor arma contra el *malware* que nos acecha día a día.

Referencias

[1] ¿Qué hace un antivirus para detectar el malware?. INCIBE. (2020). Recuperado el 10 de Marzo de 2020, desde https://www.incibe.es/protege-tu-empresa/blog/hace-antivirus-detectar-el-malware.

[2] Pastorino, C. (2020). Hecha la ley, hecha la trampa: ¿alcanza con el antivirus? — WeLiveSecurity. WeLiveSecurity. Recuperado el 7 de Abril de 2020, desde https://www.welivesecurity.com/la-es/2017/01/12/alcanza-el-antivirus/.

[3] Sihwail, R., Omar, K., & Zainol Ariffin, K. (2018). A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis. International Journal On Advanced Science, Engineering And Information Technology, 8(4-2), 1662. https://doi.org/10.18517/ijaseit.8.4-2.6827

[4] Hu, W., & Tan, Y. (2017). Generating adversarial malware examples for black-box attacks based on gan. ArXiv:1702.05983 [Cs]. http://arxiv.org/abs/1702.05983

[5] Anderson, H. S., & Roth, P. (2018). Ember: An open dataset for training static pe malware machine learning models. ArXiv:1804.04637 [Cs]. http://arxiv.org/abs/1804.04637

[6] Chen, B., Ren, Z., Yu, C., Hussain, I., & Liu, J. (2019). Adversarial examples for cnn-based malware detectors. IEEE Access, 7, 54360–54371. https://doi.org/10.1109/ACCESS.2019.2913439

[7] He, K., & Kim, D.-S. (2019). Malware detection with malware images using deep learning techniques. 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 95–102. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00022

[8] Malware Detection Using Deep Learning. Medium. (2020). Recuperado el 5 de Marzo de 2020, desde https://towardsdatascience.com/malware-detection-using-deep-learning-6c95dd235432.

[9] Rathore, H., Agarwal, S., Sahay, S. K., & Sewak, M. (2018). Malware detection using machine learning and deep learning. ArXiv:1904.02441 [Cs], 11297, 402–411. https://doi.org/10.1007/978-3-030-04780-1_28

[10] Performance Metrics for Classification problems in Machine Learning. Medium. (2020). Recuperado el 8 de Marzo de 2020, desde https://medium.com/thalus-ai/performance-metrics-for-classification- problems-in-machine-learning-part-i-b085d432082b.

[11] Kukhnavets, P. (2020). How Gantt Charts Simplify and Empower Project Management. Habr.com. Recuperado el 16 de Marzo de 2020, desde https://habr.com/en/company/hygger/blog/461835/.

[12] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., & Nicholas, C. (2017). Malware detection by eating a whole exe. ArXiv:1710.09435 [Cs, Stat]. http://arxiv.org/abs/1710.09435

[13] Nikhilesh Kasturi, S. (2020). LightGBM vs XGBOOST: Which algorithm win the race !!!. Medium. Recuperado el 26 de Marzo de 2020, desde https://towardsdatascience.com/lightgbm-vs-xgboost-which-algorithm-win-the-race-1ff7dd4917d.

[14] Researchers Easily Trick Cylance's AI-Based Antivirus Into Thinking Malware Is 'Goodware'. Vice. (2020). Recuperado el 4 de Febrero de 2020, desde https://www.vice.com/en_us/article/9kxp83/researchers-easily-trick-cylances-ai-based-antivirus-into-thinking-malware-is-goodware.

[15] Grosse, K., Papernot, N., Manoharan, P., Backes, M., & McDaniel, P. (2017). Adversarial examples for malware detection. In S. N. Foley, D. Gollmann, & E. Snekkenes (Eds.), Computer Security – ESORICS 2017 (Vol. 10493, pp. 62–79). Springer International Publishing. https://doi.org/10.1007/978-3-319-66399-9_4

[16] Media.kaspersky.com. (2020). Recuperado el 16 de Febrero de 2020, desde https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf.

[17] Sarkar, P. (2020). Bagging and Random Forest in Machine Learning. KnowledgeHut Blog. Recuperado el 8 de Abril de 2020, desde https://www.knowledgehut.com/blog/data-science/bagging-and-random-forest-in-machine-learning.

[18] Ke, Guolin and Meng, Qi and Finely, Thomas and Wang, Taifeng and Chen, Wei and Ma, Weidong and Ye, Qiwei & Liu, Tie-Yan (2017) LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Microsoft. Recuperado el 1 de Mayo de 2020, desde https://www.microsoft.com/en-us/research/publication/lightgbm-a-highly-efficient-gradient-boosting-decision-tree/.

[19] Rocca, J. (2020). Ensemble methods: bagging, boosting and stacking. Medium. Recuperado el 15 de Abril de 2020, desde https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205.

[20] Kho, J. (2020). Why Random Forest is My Favorite Machine Learning Model. Medium. Recuperado el 30 de Abril del 2020, desde https://towardsdatascience.com/why-random-forest-is-my-favorite-machine-learning-model-b97651fa3706.

[21] Pandas documentation — pandas 1.0.3 documentation. Pandas.pydata.org. (2020). Recuperado el 9 de Mayo del 2020, desde https://pandas.pydata.org/docs/.

[22] Vijay, U. (2020). Early Stopping to avoid overfitting in neural network- Keras. Medium. Recuperado el 5 de Mayo del 2020, desde https://medium.com/@upendravijay2/early-stopping-to-avoid-overfitting-in-neural-network-keras-b68c96ed05d9.

[23] Cloud Computing Services — Google Cloud. Google Cloud. (2020). Recuperado el 5 de Mazo del 2020, desde https://cloud.google.com/.

[24] Domínguez, P. (2020). En qué consiste el modelo en cascada. OpenClassrooms. Recuperado el 4 de Marzo del 2020, desde https://openclassrooms.com/en/courses/4309151-gestiona-tu-proyecto-de-desarrollo/4538221-en-que-consiste-el-modelo-en-cascada.