UOC Universitat Oberta de Catalunya

UAB Universitat Autònoma de Barcelona

UNIVERSITAT ROVIRA i VIRGILI

UNIVERSITAT OBERTA DE CATALUNYA

UNIVERSITAT AUTÓNOMA DE BARCELONA

UNIVERSITAT ROVIRA I VIRGILI

MÁSTER UNIVERSITARIO EN SEGURIDAD DE LAS TECNOLOGÍAS DE LA INFORMACIÓN Y DE LAS COMUNICACIONES

TRABAJO FIN DE MÁSTER

ANCERT: APLICACIÓN DE TÉCNICAS DE MACHINE LEARNING A LA SEGURIDAD

Isaac Moles Buyo

Director: Enric Hernández

Resumen

Stephen Hawking, uno de los más eminentes científicos, predijo que la inteligencia artificial podía suponer el fin de la raza humana, lo cual da una escabrosa idea del poder intrínseco que ésta puede llegar a contener. A pesar de que nos encontramos en las primeras etapas de que ciertas formas de inteligencia artificial sean ya una realidad, como por ejemplo el Machine Learning, resultan ya sorprendentes las posibilidades que nos brinda, permitiendo afrontar problemas hasta hace poco inabarcables. En este Trabajo Final de Máster, nos hemos centrado en el problema de identificar mediante un algoritmo de Machine Learning, cuando nos encontramos ante un ataque de Denegación de Servicio (DoS) o un ataque distribuido de Denegación de Servicio (DDoS). Nos hemos decantado por el algoritmo de random forest, por su buen rendimiento y no excesiva complejidad.

Abstract

Stephen Hawking, one of the most eminent scientists warned artificial intelligence could end mankind, which gives us an idea of how much powerful may become. Although we are just in the earlier stages of certain artificial intelligence, such as Machine Learning, the possibilities that they bring us are amazing already, allowing us to face new challenges. In this Final Dissertation, we have focused on detecting through a Machine Learning algorithm when a Deny of Service (DoS) or Distributed Deny of Service (DDoS) attack is taking place. Between all the possibilities we have chosen the random forest algorithm, due to its great performance and simplicity.

Agradecimientos

Quisiera dedicar este trabajo a varias personas que por varios motivos me aportan un gran apoyo en mi vida.

En primer lugar a Enric, mi director de este trabajo, por su gran paciencia y por estar disponible siempre que lo he necesitado.

A mis compañeros y profesores del máster MISTIC, por compartir su conocimiento y la pasión por la seguridad informática.

A mi familia, tanto de España como de Corea, por su cariño incondicional y apoyarme en todos mis proyectos.

A mi mujer, por compartir conmigo cada día y ser mi alma gemela.

사랑합니다

Tabla de contenido

Índice de ilustraciones

Introducción

Cada vez resulta más evidente que la Inteligencia Artificial (IA) constituye uno de los fundamentos sobre los que se construirán las tecnologías del futuro. En un mundo cada vez más interconectado y globalizado, el volumen de datos que se generan es tan ingente que resulta imposible de gestionar y analizar por técnicas tradicionales, es el llamado Big Data, que constituye en esencia el combustible que alimenta a la inteligencia artificial.

Entre los múltiples ámbitos en los que ya se está aplicando la IA, abriendo horizontes hasta ahora inalcanzables, encontramos las llamadas *smart cities*, el reconocimiento facial adaptativo, la predicción de conductas de mercado o la seguridad informática. Es precisamente en este último campo en el que se sitúa este trabajo final de máster, y más en concreto en la detección temprana mediante técnicas basadas en *Machine Learning* (ML) de dos de los más comunes y efectivos ataques: los ataques de Denegación de Servicio (DoS) y los ataques Distribuidos de Denegación de Servicios (DDoS). Estos ataques son especialmente peligrosos, principalmente por el bajo coste que comportan para el ejecutor pero las grandes pérdidas que ocasiona a la víctima: debido a la proliferación de *botnets* se estima que el precio de contratar durante una hora el uso de un millón de equipos está alrededor de 10 dólares, y de unos 150 dólares la contratación para un ataque de una semana de duración[1], mientras que para la víctima tienen un coste promedio de 40.000 dólares por hora.

A pesar de que la más novedosa y sofisticada expresión de la IA la encontramos en el Deep Learning (DL) y las arquitecturas de aprendizaje profundo como las redes neuronales profundas, en este Trabajo Final de Máster nos hemos enfocado en ML y en concreto hemos utilizado un algoritmo de *random forest*, principalmente para poder comprobar si mediante un algoritmo de complejidad media y de bajo coste computacional, es posible defenderse de una forma efectiva contra esos costosos ataques.

[1] (Olleros & Zhegu, 2016, pág. 434)

Objetivo

El principal objetivo de este trabajo era demostrar si basándonos en un algoritmo de complejidad media-baja como es el caso de *random forest* es posible afrontar problemas de gran complejidad como es la detección de ataques de denegación de servicio, ya sean éstos distribuidos o no.

También formaba parte de nuestro objetivo desarrollar nuestro algoritmo utilizando un lenguaje de uso extendido como es Python, fácilmente portable y ejecutable desde multitud de sistemas operativos.

Finalmente, pero no menos importante, con la realización de este Trabajo se perseguía también recopilar valiosa teoría sobre los fundamentos de la inteligencia artificial, clarificar conceptos como Machine Learning y Deep Learning y documentación del estado del arte en que se encuentra la inteligencia artificial, así como los problemas que ya se están afrontando mediante tecnologías basadas en ello.

Enfoque

El desarrollo de este Trabajo se enfocó en el alcance de diversas metas parciales distribuidas a lo largo del semestre. Estas etapas eran las siguientes:

-Primera etapa: consistía en documentarse acerca de todos los conceptos fundamentales de la Inteligencia Artificial y distinguir entre las tecnologías de Machine Learning y Deep Learning. También incluía analizar los principales algoritmos disponibles basados en Machine Learning y estudiar algunos casos reales en los que ya se están aplicando.

-Segunda etapa: consistía en estudiar situaciones en las que se puede aplicar Machine Learning para intentar resolver de forma eficiente un problema complejo.

-Tercera etapa: finalmente se pretendería desarrollar un algoritmo basado en Machine Learning que permitiese dar solución a alguno de los problemas detectados en la etapa segunda.

Trabajos existentes en este ámbito

A pesar de tratarse de una materia de reciente desarrollo, el Machine Learning aplicado a la seguridad informática ha experimentado en los últimos años un gran desarrollo y se han desarrollado algunos trabajos en este sentido. En el caso concreto que ocupa a este TFM, se ha encontrado que un anterior estudiante de este Máster abarcó un tema muy parecido[2], y

[2] (Merchán Macías)

además utilizó el mismo dataset que se decidió utilizar para este trabajo. Por otra parte, ese trabajo no se enfocó en un único algoritmo si no que realizó una comparativa entre múltiples, e implementó todo su código con lenguaje R.

Este TFM ha intentado aportar un valor recopilatorio al tema del Machine Learning aplicado a la seguridad informática, y desarrollar un caso práctico mediante uno de los datasets de mejor calidad, haciéndolo íntegramente en Python, al ser éste uno de los lenguajes más extendidos y polivalentes.

Planificación del trabajo

Las etapas indicadas en el apartado anterior se fijaron para que coincidieran de forma aproximada con las fechas de entrega de cada una de las PAC o Pruebas de Avaluación Continuada:

08/10/2018: Primera etapa

09/11/2018: Segunda etapa

03/12/2018: Tercera etapa

Se estimó el tiempo necesario para la realización de este TFM en unas 225horas (9 créditos ECTS * 25horas), por lo que se planificó dedicarle unas 17,5horas/semana, al disponer de cerca de 13 semanas.

Equipo utilizado

Para realizar este TFM se ha utilizado un ordenador portátil de gama media con las siguientes características:

Procesador i5 4.200U @1.60GHz
8GB de memoria RAM
Sistema operativo de 64 bits (Windows 10 Pro)
Tarjeta gráfica Nvidia GeForce GT 740M

Debido a la escasa capacidad de proceso, el set de pruebas de ejecución no es lo amplio que se hubiera deseado. En concreto para nuestro algoritmo *random forest* hubiese sido especialmente interesante disponer de una tarjeta gráfica con muchos hilos de proceso, ya que de esta forma hubiera podido sacar arrojar mucho mejor rendimiento, al poder procesar de forma simultánea muchos árboles del bosque.

Organización de esta memoria

Este documento se encuentra estructurado en los siguientes capítulos:

-Capítulo 1: contiene una breve introducción a este Trabajo Final de Máster, donde se recogen las motivaciones que lo justifican, qué aporta este TFM y como se ha estructurado su desarrollo.

-Capítulo 2: se recogen algunos conceptos básicos de Machine Learning, así como los principales algoritmos disponibles. También presentamos una visión del actual estado del arte en lo que respecta a aplicaciones existentes basadas en Machine Learning dándole un especial foco al campo de la seguridad informática.

-Capítulo 3: presentamos los ataques DoS y DDoS que serán sobre los que desarrollaremos nuestro algoritmo.

Capítulo 4: presentamos las diferentes partes que componen nuestro algoritmo.

Capítulo 5: conclusiones extraídas de la realización de este TFM.

Conceptos básicos

Principales técnicas y algoritmos de Machine Learning

Los algoritmos de ML se pueden clasificar en función de la técnica en que están basados[3] dependiendo de la estrategia que utilicen para extraer conocimiento del dataset:

-Aprendizaje supervisado: en este caso adopta especial importancia el rol de maestro o supervisor, que a la práctica y en los algoritmos actuales es desempeñado por conjuntos de datos de entrenamiento que están formados por pares de datos (una entrada y la salida esperada correspondiente). No obstante, es necesario que el algoritmo sea lo suficientemente flexible para no depender en exceso de dichos datos de entrenamiento, o en caso contrario nos encontraremos con el inconveniente denominado sobre-ajuste.

Uno de los principales algoritmos de este grupo es el de regresión lineal, el cuál intenta encontrar una línea que represente la nube de puntos que corresponderían a los distintos valores que componen nuestro dataset. El principal inconveniente que presenta este algoritmo es "overfit"[4], es decir, el ajuste excesivo a los datos disponibles, con el consecuente riesgo de no ofrecer buenos resultados para nuevos valores.

-Aprendizaje no supervisado: este grupo está formado por algoritmos cuya estrategia consiste en identificar rasgos que permitan segregar los datos dependiendo del valor de esos rasgos. Por ejemplo sería el caso de un algoritmo que dependiendo de la opinión vertida por los usuarios acerca de determinados libros, emita recomendaciones de libros no solapados entre usuarios que presenten analogías en sus gustos.

-Aprendizaje semi-supervisado: hay multitud de situaciones en las que no es factible adoptar ninguna de las dos estrategias que acabamos de presentar, debido a que el dataset utilizado para entrenar el algoritmo no resulta representativo del dominio completo en el que se mueven los datos, por ese motivo es necesario encontrar un punto intermedio entre ambos enfoques.

-Aprendizaje reforzado: Incluso si no utilizamos el rol de supervisor, es posible entrenar nuestro algoritmo mediante el envío de recompensas o penalizaciones. La estrategia del aprendizaje reforzado se basa en la idea de que un agente racional siempre intenta lograr aquellos objetivos que le aportan beneficios.

-Neurociencia computacional: Este último grupo de algoritmos está estrechamente ligado a la evolución de la neurociencia, campo que desde que Santiago Ramón y Cajal junto a Camillo Golgi alrededor del 1900 descubriesen la estructura de las neuronas (obteniendo conjuntamente el premio Nobel por ello[5]) no ha parado de desarrollarse de forma cada vez

[3] (Bonaccorso, 2018, págs. 14-23)
[4] (Yan, 2009, pág. 157)
[5] (Swanson, 2017, pág. 11)

más rápida. De esta forma, estos algoritmos tratan de emular el funcionamiento del cerebro, siendo el Deep Learning la vertiente más sofisticada del Machine Learning.

Por otra parte, cabe indicar que los problemas a los que enfrentamos nuestros algoritmos se suelen clasificar en dos grandes grupos: problemas de regresión o problemas de clasificación. Para entender de forma sencilla en qué consiste cada grupo, podemos considerar que los primeros tratan con datos que pueden adoptar un rango de valores continuo y difuso, mientras que los segundos se pueden agrupar de forma discreta en distintos grupos de valores delimitados. Un problema de regresión podría ser calcular el precio al que se venderá un determinado producto (valor continuo), mientras que un problema de clasificación podría ser etiquetar si una determinada imagen se corresponde con un perro o un gato.

Casos reales en los que se está aplicando Machine Learning

Uno de los ámbitos en los que más aplicación se está dando a las tecnologías basadas en Inteligencia Artificial es el de las *smart cities*, dónde cada vez más se está haciendo necesario disponer de tecnologías que permitan el procesado de datos de forma desatendida, debido al ingente volumen de información que se recibe desde los sensores que se van desplegando. Veamos a continuación algunos casos prácticos en los que ya se está aplicando Machine Learning:

-Regulación del tráfico: Uno de los principales retos a los que se enfrentan las metrópolis es el de la reducción del tiempo necesario para desplazarse entre dos puntos, por un lado debido a los atascos de tráfico que tienen lugar y por otra parte también debido al tiempo que es necesario invertir para encontrar aparcamiento. Ambos problemas son susceptibles de optimizarse mediante la aplicación de tecnologías basadas en Machine Learning. Por ejemplo, en la ciudad de San Diego[6] tras instalar señales de control de tráfico inteligentes[6] han obtenido una reducción de hasta el 25% en la duración de los trayectos, y las detenciones de los vehículos se visto disminuidas en un 53% durante las horas punta.

-Incrementar la seguridad pública: Otra aplicación basada en Machine Learning que ya estamos viendo en grandes ciudades como Nueva York, Seattle o Nueva Orleans[7] es la utilización del análisis de big data para reducir el nombre de accidentes de tráfico, identificando en qué intersecciones tienen lugar y cómo se producen.

Si nos centramos en el ámbito de la seguridad informática, podemos observar que la irrupción de la inteligencia artificial ha revolucionado el concepto de EDR[8] y se ha incorporado a múltiples opciones que ya hay disponibles hoy en día en el mercado, de la mano de los principales fabricantes, y que ya gozan de amplio reconocimiento entre los profesionales del sector de la seguridad. Podemos destacar los siguientes:

-Palo Alto Traps[9]: esta herramienta utiliza el machine learning para detectar de forma precoz comportamientos que corresponden a software malicioso, permitiendo bloquearlo en las primeras etapas de su explotación y de esta forma neutralizando los posibles daños antes que los llegue a causar. Para ello se basa en la idea de que a pesar de existir infinitud de malware, la práctica totalidad de ellos se basan en un número muy limitado de estrategias para ejecutar su actividad maliciosa.

-Symantec Endpoint Protection[10]: de forma parecida a Traps, esta herramienta se basa en técnicas de machine learning para detectar en tiempo de ejecución la presencia de software malintencionado que intenta ejecutarse en el equipo, y lo hace detectando comportamientos que se salen de lo que se considera funcionamiento normal del sistema operativo. Por ejemplo

[6] (Faulconer, 2018)
[7] (Vision Zero Labs: Using Data Science to Improve Traffic Safety)
[8] (Singh Chauhan, 2018, pág. 139)
[9] https://www.paloaltonetworks.com/products/secure-the-endpoint/traps
[10] https://www.symantec.com/products/endpoint-protection

si detecta que un determinado registro del sistema operativo está siendo modificado sin que se hayan solicitado por el usuario cambios que pudiesen justificarlo.

-McAfee Endpoint Security[11]: el mismo principio que en las anteriores herramientas. La aplicación de técnicas basadas en Machine Learning permite a esta herramienta, de forma parecida a las anteriores, bloquear incluso ataques de día cero, al no basarse en la detección de firmas como hacían los antivirus tradicionales, sino en patrones de comportamiento.

-Sophos Intercept X[12]: en este caso el fabricante británico indica que su solución utiliza la potencia del Deep Learning para, combinándola con su tecnología anti-exploit, neutralizar la más amplia variedad de amenazas. En concreto, Sophos indica hacer uso de una red neuronal, remarcando que aunque esta estrategia requiere de ingentes cantidades de datos y poder computacional, es capaz de arrojar mejores resultados que otras técnicas de Machine Learning como *random forest*, *k-means clustering* o redes Bayesianas.

[11] (McAfee, 2017)
[12] (Sophos, 2018)

Ataques de denegación de servicio (DoS) y ataques distribuidos de denegación de servicio (DDoS)

El objetivo que persiguen tanto los ataques de denegación de servicio (DoS) como los ataques distribuidos de denegación de servicio (DDoS) es, como su nombre indica, el de ocasionar un corte en el normal funcionamiento de un determinado servicio, normalmente mediante la saturación por medio de peticiones maliciosas que llegan a colapsar los sistemas informáticos del prestador de ese servicio. La principal diferencia[13] entre ambos ataques es que mientras en el ataque DoS éste proviene de un único equipo, el DDoS proviene de varios o incluso miles equipos, normalmente *zombis* que forman parte de lo que se llama *botnet*.

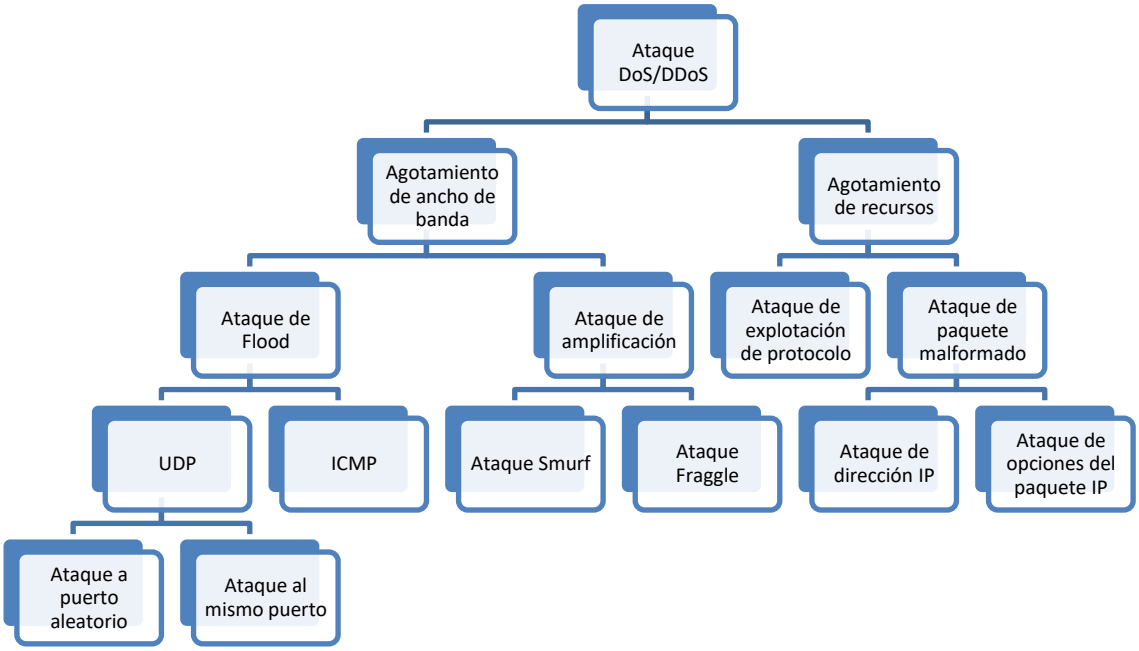


Ilustración 1. Taxonomía de ataques DoS y DDoS. Mirkovic and Reiher. (2004)

Tal como podemos ver en la Ilustración 1, existen un gran número de modalidades de este tipo de ataques, siendo el más común el de *SYN flood*[14]. Éste consiste en que el atacante envía múltiples peticiones de conexión (SYN) a la víctima, que responde a su vez con otro SYN y al mismo tiempo crea una conexión medio abierta (*half-open connection*) a la espera de recibir el ACK por parte del solicitante. En este caso al tratarse del atacante, dicho ACK nunca será recibido, y por tanto la *half-open connection* será mantenida, consumiendo ciclos de CPU y memoria. Si este proceso es repetido muchas veces en un plazo muy corto de tiempo por parte del atacante, la víctima llegará eventualmente a consumir todos sus recursos y tendrá lugar la denegación de servicio.

[13] (Sud & Edelman, 2004, pág. 17)
[14] (Mason & Newcomb, 2001, pág. 28)

En la Ilustración 2 podemos observar el funcionamiento de una conexión TCP normal:

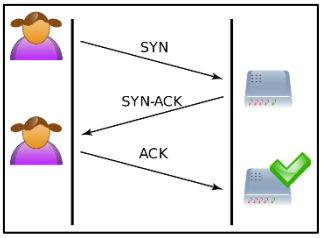


Ilustración 2 Conexión TCP normal

En la Ilustración 3 por el contrario vemos el funcionamiento de un ataque *SYN flood*, y cómo la víctima no es capaz de atender posteriormente una petición legítima por parte de un cliente.

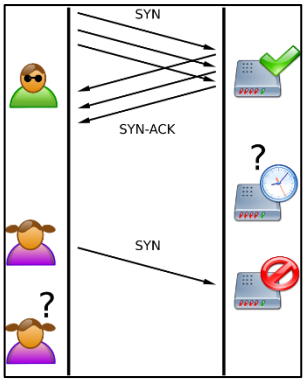


Ilustración 3 Ataque SYN flood

Este tipo de ataque puede clasificarse en tres modalidades[15]:

1) Ataque directo: el atacante envía multitud de peticiones SYN sin falsear su dirección IP, y configurando su sistema operativo o su firewall para que no responda a las SYN ACK que reciba de la víctima.
2) Ataque distribuido: en este caso el atacante dispone de múltiples máquinas o de una *botnet* y perpetra el ataque enviando las peticiones SYN desde todas ellas. Ésta es tal vez la versión más difícil de neutralizar.
3) Ataque falseando el origen. En esta modalidad el atacante envía las peticiones SYN desde direcciones falsas, de modo que cuando la víctima intente responder se encontrará con que no recibe respuesta, al ser falsas esas direcciones IP.

El objetivo final de nuestro algoritmo de ML es ser capaces de detectar de forma prematura cuándo está teniendo lugar un ataque de DoS o DDoS. Para ello, primero de todo deberemos disponer de un dataset apropiado e identificar qué aspectos de esos datos nos resultan relevantes para poder llevar a cabo la detección.

[15] (Kamila, 2017, pág. 301)

Desarrollo de nuestro algoritmo

Selección del dataset

Cuando nos planteamos el desarrollo de un proyecto basado en ML uno de los aspectos que adopta una importancia primordial es el de poder nutrir nuestros algoritmos con un dataset tan amplio como sea posible pero al mismo tiempo que los datos de éste sean de gran calidad. Esto se consigue únicamente mediante un minucioso proceso de saneamiento de los datos, proceso que llega a consumir hasta el 90 por ciento[16] del tiempo de los científicos de datos.

Como tendremos ocasión de ver más atrás, existen gran variedad de algoritmos de ML y cada uno de ellos requiere un tipo de dataset distinto para su entrenamiento. Así, por ejemplo, un algoritmo supervisado requerirá de un dataset compuesto de pares de datos (entrada y resultado esperado) que actuará a modo de entrenador o supervisor, permitiéndole corregir su error de una forma progresiva y que al mismo tiempo no lo conduzca a una dependencia excesiva del dataset, lo que se llama sobreajuste o *overfitting*, y que ocasiona que el algoritmo arroje errores excesivos cuando se enfrenta a datos nuevos, que no había encontrado durante su entrenamiento con dataset.

Por el contrario, ante un algoritmo no supervisado deberemos partir de la base que no existirá ese rol de entrenador y por lo tanto no será posible determinar el error absoluto. En este caso el dataset deberá incluir de forma implícita ciertas características que permitan al algoritmo agrupar los datos en función de determinadas características, siendo ésta una de las principales aplicaciones de este tipo de algoritmos.

Si nos encontramos con un dataset que resulta limitado, por su tamaño o variedad de datos, posiblemente nos decantemos por un algoritmo de tipo semi-supervisado, ya que en este tipo de algoritmos el dataset suele abarcar una fracción reducida del rango total de los datos posibles y el algoritmo está diseñado para intentar suplir esa reducida muestra, por ejemplo propagando etiquetas de elementos identificados hacia aquellos que no lo son o utilizando Máquinas de Soporte de Vectores (SVM)[17].

En el caso de los algoritmos de aprendizaje reforzado, es el entorno el que ofrece el "feedback" en forma de recompensas o penalizaciones dependiendo del nivel de acierto del algoritmo, por tanto el dataset que se desee utilizar debería contener valores propios del entorno en el que vaya a ponerse en funcionamiento ese algoritmo.

Finalmente, en el caso de los algoritmos basados en redes neuronales, si nos encontramos ante una red neuronal de una única capa, el proceso de entrenamiento será parecido a algoritmos convencionales como los de aprendizaje reforzado, ya que el valor de las penalizaciones en el caso de error puede ser calculado de forma directa en función de los pesos, pero si nos encontramos ante una red neuronal de múltiples capas será un proceso más

[16] (Miller, 2017, pág. 51)
[17] (Christmann, 2008, pág. 7)

complejo al ser necesario para obtener el valor de la penalización ponderar los pesos de cada una de las capas, cuyo cálculo requiere de una compleja función compuesta [18].

Datasets de DoS y DDoS

De forma similar a como sucede para cualquier otro ámbito, en el desarrollo de aplicaciones basadas en ML es fundamental disponer de un amplio y diverso dataset que abarque la mayor variedad posible de ataques y patrones de comportamiento.

En el concreto caso que nos ocupa en este TFM, los ataques DoS y DDoS, se da la particularidad que continuamente aumentan su complejidad y sofisticación para lograr eludir los sistemas de detección de intrusiones (IDS), además de resultar extremadamente compleja y costosa su reproducción en un entorno real. Por estos motivos los investigadores han desarrollado algoritmos y mecanismos para simularlos[19].

Dataset	Autor	Fecha	Real o Simulado	Características	Tipo ataque DDoS	Tamaño	Disponibilidad	Ventajas	Limitaciones
KDD'99 Cup[20]	MIT Lincoln Labs		Simulado	-2 semanas sin ataque y cinco con ataques. -38 tipos de ataques	SYN flood	743 MB	Disponible	-Fácil de obtener -Muchos tipos de ataques	-Contenido sin balanceo conteniendo 80% tráfico de ataque
DoS_traces-20020629	University of Southern California – Information Sciences Institute	Jun 29, 2002 to Aug 14, 2002	Real	-Series temporales de 80 ataques DoS con 1 milisegundo de granularidad	Reflector attack TCP-no flag attack IP-proto 255 attack	4.1 GB	Restringido		
CAIDA DDoS Attack 2007	Paul Hick	Aug 4, 2007	Simulado	-1 hora de datos anonimizados -Consumo de recursos	UDP flood	21 GB	Semi-restringido	-Disponible para uso público -Efectivo para entrenar ataques DDoS > 5 GB	
FRGPNTP Flow Data-20131201	Colorado State University	Dec 01, 2013 to Feb 28, 2014	Anonymized	3 months daily NTP in Argus flow on 10 Gb/s link	NTP reflection attack	3.5 TB	Restringido		
EPA http Dataset	Laura Bottomley	Aug 29, 1995	Real	-46,014 GET requests - 1622 POST requests -107 HEAD requests -6 invalid requests -One-second accuracy on timestamp	HTTP flooding	4.4 MB	Disponible	-Tamaño de dataset pequeño	-No se puede determinar si las peticiones HTTP son legítimas o no. -Su reducido tamaño puede reducir su efectividad

[18] (Aggarwal, 2018, pág. 21)
[19] (Alzahrani & Hong, 2018)
[20] (Özgür & Erdem, 2016)

Recolección de datos y dataset utilizado

Una de las partes fundamentales del diseño de un algoritmo basado en ML es la de recolección de datos, que serán los que posteriormente permitirán entrenar el algoritmo. En esta fase un concepto que adquiere gran relevancia es el de entropía[21], que podemos definir como el nivel de desorden existente en un sistema. Durante la recolección de datos podemos identificar tres aspectos fundamentales, algunos de los cuales será interesante que tengan una entropía alta y otros la tengan baja:

-Origen: es conveniente que sea lo más alta posible, ya que significará que nuestros datos están siendo recopilados de una amplia variedad de fuentes, aportándole un mayor realismo y cobertura.

-Lugar de almacenamiento: en este caso será conveniente minimizar su entropía, para de esta forma simplificar la unificación de todas las fuentes que nos aportan los datos. Por ejemplo si estamos almacenando por separado los datos del firewall, los datos del IDS y los datos del servidor web, la tarea de unificación se antojará ardua y compleja.

-Valor: esta característica también será interesante que tenga una entropía baja, con el fin de evitar datos que se escapan de nuestros objetivos y que al fin y al cabo únicamente nos estarían aportando ruido si los incorporamos.

El proceso de recolección de datos para elaborar un dataset de calidad se escapaba del alcance de este trabajo, puesto que el poder "capturar" cuando tiene lugar un ataque DoS o DDoS requiere de un largo periodo de recolección de *logs* y disponer de una infraestructura que resulte representativa de una corporación medianamente equipada (*firewalls*, *proxies*, tal vez algún *waf* y servidores de bases de datos y/o web).

Por ese motivo para nuestro algoritmo hemos decidido utilizar uno de los mejores datasets disponibles en lo que respeta a ataques SYN, el KDD-CUP 99. Cabe mencionar que este dataset ya fue utilizado en *The Third International Knowledge Discovery and Data Mining Tools Competition*[22].

Cada registro de los que contiene tiene la siguiente forma:

0,tcp,http,SF,215,45076,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,0,0,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00,normal

En la Tabla 1 tenemos una descripción de las características básicas para cada conexión TCP. Cabe destacar por ejemplo la duración de la conexión, el protocolo utilizado, el servicio en el destino o si la dirección de origen y destino son o no la misma. Algunas de estas características son de tipo continuo, mientras que otras son de tipo discreto.

[21] (Kumar G., 2019, pág. 9)
[22] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

Característica	***Descripción***	***Tipo***
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous
urgent	number of urgent packets	continuous

Tabla 1 Características básicas de cada conexión TCP

Por otra parte, en la Tabla 2 tenemos las características correspondientes a cada conexión, basadas en el conocimiento del dominio. Podemos observar que entre ellas se encuentran el número de intentos fallidos de login, si el atacante ha intentado alcanzar derechos de *root* y si ha tenido éxito o no.

Característica	***Descripción***	***Tipo***
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_hot_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete

Tabla 2 Características de cada conexión en base al conocimiento del dominio

Finalmente en la Tabla 3 tenemos las características correspondientes al tráfico, cuando aplicamos una ventana de dos segundos. Como podemos apreciar, algunas de ellas solo se aplican para conexiones cuyo origen y destino está en la misma dirección, y otras solo se aplican a aquellas conexiones con el mismo servicio.

Característica	***Descripción***	***Tipo***
count	number of connections to the same host as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-host connections.	
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
	Note: The following features refer to these same-service	

	connections.	
srv_serror_rate	% of connections that have ``SYN'' errors	Continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	Continuous
srv_diff_host_rate	% of connections to different hosts	continuous

Tabla 3 Características de tráfico obtenidas con una ventana de dos segundos

Identificación de rasgos relevantes para la identificación de los ataques

Uno de los aspectos fundamentales cuando se diseña un algoritmo basado en ML es la identificación de las características que resultan relevantes para el objetivo perseguido. En nuestro caso debemos identificar qué campos del dataset aportan mayor valor para permitir detectar que nos encontramos ante un ataque DoS o DDoS. El dataset que utilizamos contiene al final de cada registro el nombre del ataque del que se trata, o en caso de no corresponder a ningún ataque se indica con la palabra **normal.**

Pasemos a ver las características de los ataques que podemos encontrar en nuestro dataset:

- **Neptune**: Para iniciar un ataque de este tipo, un gran número de peticiones SYN son enviadas por parte del atacante, para a continuación no responder a ninguna de las respuestas que retorne el servidor víctima. A continuación podemos ver algunos registros extraídos de nuestro dataset que constituyen ejemplos de este caso:

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,112,16,1.00,1.00,0.00,0.00,0.14,0.07,0.00,255,16,0.06,0.08,0.00,0.00,1.00,1.00,0.00,0.00,neptune.

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,121,5,1.00,1.00,0.00,0.00,0.04,0.07,0.00,255,5,0.02,0.08,0.00,0.00,1.00,1.00,0.00,0.00,neptune.

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,118,3,1.00,1.00,0.00,0.00,0.03,0.07,0.00,255,3,0.01,0.08,0.00,0.00,1.00,1.00,0.00,0.00,neptune.

0,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,140,10,1.00,1.00,0.00,0.00,0.07,0.06,0.00,255,10,0.04,0.07,0.00,0.00,1.00,1.00,0.00,0.00,neptune.

- **Smurf**: En el caso de este ataque, apreciamos en el dataset que las conexiones utilizan el protocolo ICMP para solicitar peticiones de eco, y van dirigidas a la dirección de *broadcast* para originar un gran número de respuestas por parte de la víctima. Lo podemos identificar por el gran número de respuestas de eco que son enviadas desde el servidor atacado. Algunos registros de nuestro dataset que se corresponde con este tipo de ataque serían los siguientes:

0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.

0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.

0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,511,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.

0,icmp,ecr_i,SF,1032,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,510,510,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.00,1.00,0.00,0.00,0.00,0.00,0.00,smurf.

- **Teardrop**: Este tipo de ataque consiste en enviar solapamientos de fragmentos de paquetes IP, lo que ocasiona el reinicio del sistema en versiones antiguas de Windows y de Linux. Algunos ejemplos en nuestro dataset:

0,udp,private,SF,28,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,1,0.00,0.02,0.00,0.00,0.00,0.00,0.77,0.00,teardrop.

0,udp,private,SF,28,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,2,0.01,0.02,0.01,0.00,0.00,0.00,0.77,0.00,teardrop.

0,udp,private,SF,28,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,3,0.01,0.02,0.01,0.00,0.00,0.00,0.77,0.00,teardrop.

0,udp,private,SF,28,0,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,4,0.02,0.02,0.02,0.00,0.00,0.00,0.77,0.00,teardrop.

- **Pod (Ping Of Death)**: Este ataque consiste en enviar paquetes IP sobredimensionados que ocasionan un desbordamiento en el sistema operativo de la víctima, únicamente si ésta usa un sistema operativo antiguo. Encontramos varias muestras en nuestro dataset:

0,icmp,ecr_i,SF,1480,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,2,0.01,0.08,0.01,0.00,0.61,0.00,0.01,0.00,pod.

0,icmp,ecr_i,SF,1480,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,3,0.01,0.08,0.01,0.00,0.60,0.00,0.01,0.00,pod.

0,icmp,ecr_i,SF,1480,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,4,0.02,0.08,0.02,0.00,0.60,0.00,0.01,0.00,pod.

0,icmp,ecr_i,SF,1480,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,0.00,0.00,1.00,0.00,0.00,255,5,0.02,0.08,0.02,0.00,0.60,0.00,0.01,0.00,pod.

- **LAND (Local Area Network Denial)**: Este ataque tiene cierto parecido con el de SYN flood, al enviar gran número de paquetes TCP de petición de SYN, pero se diferencia en que en este caso incluye una falsificación tanto de la dirección origen como de la destino, conteniendo en ambas la dirección de la víctima, con la intención de que cuando se ésta envíe la respuesta entre en un bucle que acabe consumiendo todos sus recursos. Hay pocos ejemplos de este ataque en nuestro dataset, pero mostramos a continuación alguno de ellos:

0,tcp,telnet,S0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1.00,1.00,0.00,0.00,1.00,0.00,0.00,1,1,1.00,0.00,1.00,0.00,1.00,1.00,0.00,0.00,land.

0,tcp,finger,S0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,1.00,1.00,0.00,0.00,1.00,0.00,1.00,15,1,0.07,0.20,0.07,0.00,0.07,1.00,0.07,0.00,land.

0,tcp,finger,S0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,2,0.33,1.00,0.67,0.00,0.33,1.00,1.00,16,2,0.12,0.19,0.12,0.00,0.12,1.00,0.06,0.00,land.

0,tcp,finger,S0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,1.00,1.00,0.00,0.00,1.00,0.00,1.00,1,1,1.00,0.00,1.00,0.00,1.00,1.00,0.00,0.00,land.

- **Back:** En este tipo de ataque la víctima reemplaza el valor del campo IP origen por una dirección falsa, de forma que la víctima es incapaz de bloquear el ataque al venirle cada vez de IPs distintas, ocasionándole que finalmente acabe consumiendo todos sus recursos y le resulte imposible atender peticiones legítimas. Algunos ejemplos en nuestro dataset:

14,tcp,http,RSTR,42340,1460,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,1.00,1.00,1.00,0.00,0.00,1,1,1.00,0.00,1.00,0.00,0.00,0.00,1.00,1.00,back.

12,tcp,http,RSTR,13140,1460,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,1.00,1.00,1.00,0.00,0.00,2,2,1.00,0.00,0.50,0.00,0.00,0.00,1.00,1.00,back.

13,tcp,http,RSTR,51100,4380,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,3,3,0.00,0.00,1.00,1.00,1.00,0.00,0.00,3,3,1.00,0.00,0.33,0.00,0.00,0.00,1.00,1.00,back.

14,tcp,http,RSTR,33580,7300,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,4,4,0.00,0.00,1.00,1.00,1.00,0.00,0.00,4,4,1.00,0.00,0.25,0.00,0.00,0.00,1.00,1.00,back.

Framework utilizado para realizar este proyecto

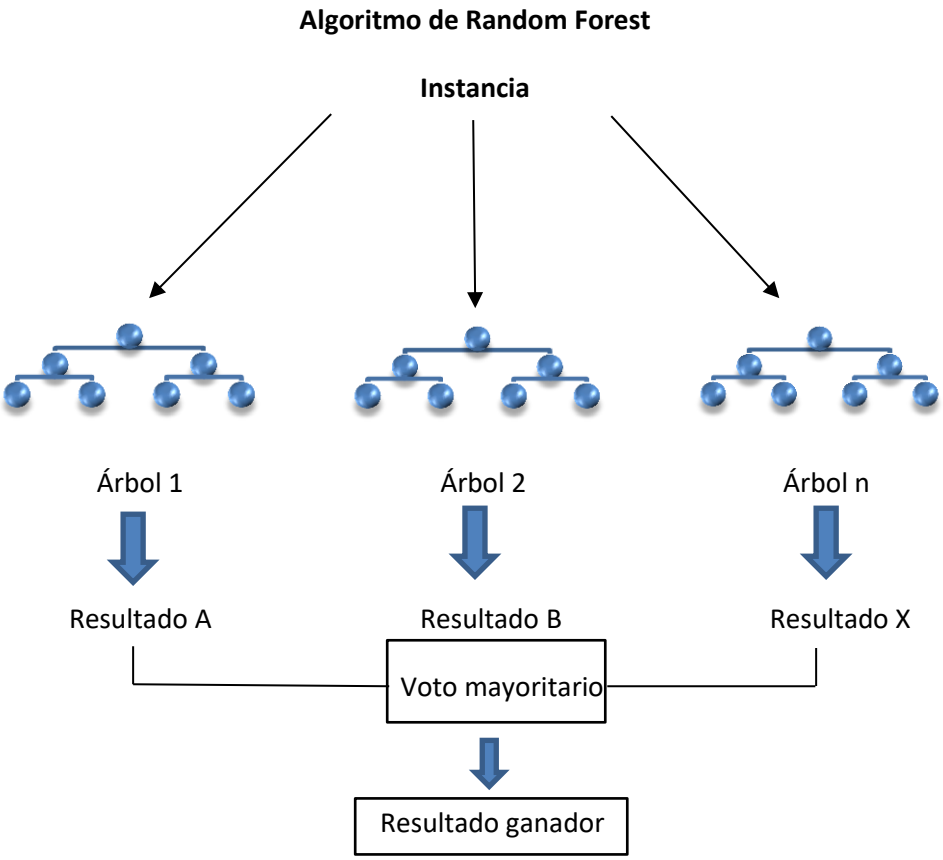
Para realizar este Trabajo Final de Máster se ha utilizado la distribución de Python que nos ofrece Anaconda, junto con la interface web Jupyter Notebook. Esta combinación nos posibilita trabajar de forma ágil y sencilla con librerías de Python, así como importar datasets ya creados.

Para el manejo de los datos hemos utilizado varias librerías de Python. Principalmente la conocida *pandas*, destinada a análisis y manipulación de datos, la Scikit-learn que contiene múltiples herramientas orientadas a trabajar con algoritmos de ML y la tensorflow que también está diseñada para trabajar con ML aunque a un más bajo nivel.

Algoritmo utilizado para realizar este proyecto

En la realización de este Trabajo Final de Máster se ha optado por utilizar un algoritmo de *random forest*, debido a que se trata de uno de los algoritmos más precisos utilizables en Machine Learning. Algunas de sus principales ventajas son[23] mayor robustez que un único árbol de decisión, rapidez para su entrenamiento, posibilidad de limitar el *overfitting* si se dota de muchos árboles de decisión aleatorios y además la posibilidad de poderse utilizar tanto para problemas de regresión como clasificación.

En el siguiente diagrama podemos ver su funcionamiento:



El algoritmo *random forest* consiste en muchos árboles de decisión[24], los cuales se crean de forma independiente mediante reemplazo[25] aleatorio de los datos del training set. A cada uno de los árboles que componen el *random forest* se le asigna una parte de los datos de entrenamiento, y podríamos decir que cada árbol "vota" por su el resultado que obtiene. Finalmente el bosque elige el resultado que ha obtenido más votos y lo marca como resultado

[23] (Cole, 2018, pág. 53)

[24] (Benítez, Escudero, Kanaan, & Masip Rodó, 2014, pág. 121) Un árbol de decisión es una forma de representar regles de clasificación inherentes a los datos, con una estructura en árbol n-ario que particiona los datos de manera recursiva. Cada rama de un árbol de decisión representa una regla que decidí entre una conjunción de valores de un atributo básico (nodos internos) o realiza una predicción de la classe (nodos terminales).

[25] (Graña, Toro, Posada, Howlett, & Lakhmi, 2012, pág. 88)

final del algoritmo. Podríamos considerar que este algoritmo guarda gran similitud con la forma de funcionamiento de un sistema democrático, en el que el resultado se obtiene tras contabilizar los votos que ha obtenido cada uno de los partidos políticos por parte de todos los ciudadanos, resultando vencedor el partido con mayor número de votos.

El pseudocódigo de clasificación que usa este algoritmo sería el siguiente[26]:

k=1
Inicializar recorrido de las muestras
PARA (cada muestra *i* en el dataset inicializado)
    Crear árbol de clasificación no podado h_k (x) para la muestra *i*
    PARA (cada nodo del árbol de clasificación h_k (x))
        Aleatoriamente obtener muestra *k* de las variables de predicción
        Elegir la mejor división entre esas variables
    FIN PARA
FIN PARA
Predecir nuevos datos combinando las predicciones de los árboles
Calcular las estadísticas e importancia de las variables

[26] (Jin, Tino, Corchado, Byrne, & Yao, 2007, pág. 865)

Creación de Training Sets y Test Sets

Mediante el siguiente código importamos la librería pandas y leemos el archivo del dataset:

```
#Importamos Pandas para manipulación de datos
import pandas as pd

#Leemos el archivo y mostramos las primeras 5 filas de datos
features = pd.read_csv('kddcup.data.corrected.csv')
features.head(5)
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_di
0	0	tcp	http	SF	162	4528	0	0	0	0	...	1	1.0	
1	0	tcp	http	SF	236	1228	0	0	0	0	...	2	1.0	
2	0	tcp	http	SF	233	2032	0	0	0	0	...	3	1.0	
3	0	tcp	http	SF	239	486	0	0	0	0	...	4	1.0	
4	0	tcp	http	SF	238	1282	0	0	0	0	...	5	1.0	

5 rows × 42 columns

Si mostramos por pantalla el aspecto del dataset obtenemos lo siguiente:

```
#Mostramos la forma que tiene nuestro dataset
print ('La forma de nuestras features es:', features.shape)
```

```
La forma de nuestras features es: (4898430, 42)
```

Por lo que respeta a la distribución de los datos de nuestro dataset, tenemos que contiene la siguiente cantidad de ataques:

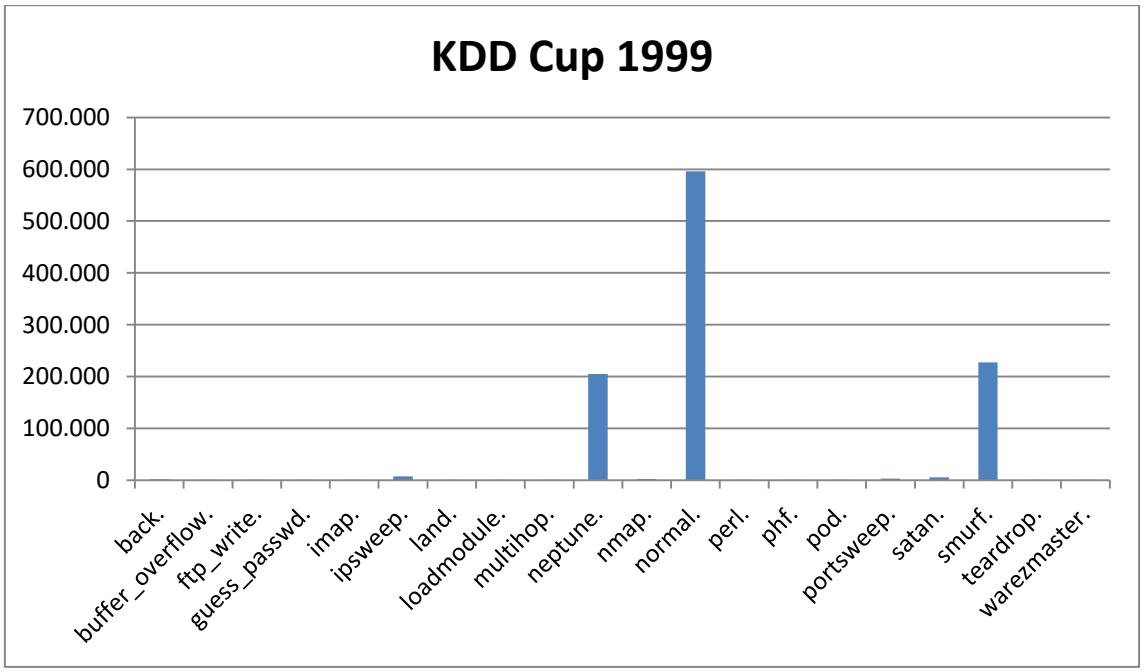


Ilustración 4 Tipos de ataques contenidos en el dataset

Podemos observar que el contenido más numeroso de nuestro dataset (595.797) corresponde con los datos normales, es decir los que corresponden a un tráfico normal de una red. Seguido por los ataques de tipo *smurf* (227.524) y los ataques *neptune* (204.815). Entre los menos presentes encontramos los ataques de *ping of death* (40).

Para entrenar nuestro algoritmo hemos destinado el 80% de los datos disponibles a *training set*, dejando el 20 % restante para comprobar su porcentaje de aciertos mediante *test set*. Con esta proporción hemos intentado encontrar el balance entre un buen nivel de entrenamiento y la disponibilidad de suficientes datos de test para poder estimar el nivel de precisión y ajustar el error que obtenemos cuando enfrentamos nuestro algoritmo con datos reales.

Esta proporción de 80/20 se encuentra dentro de lo común para este tipo de algoritmos, en los que se suele modular ligeramente dependiendo del tamaño del dataset con la finalidad de encontrar el mejor compromiso posible entre minimización del problema de varianza y problema de *bias*[27].

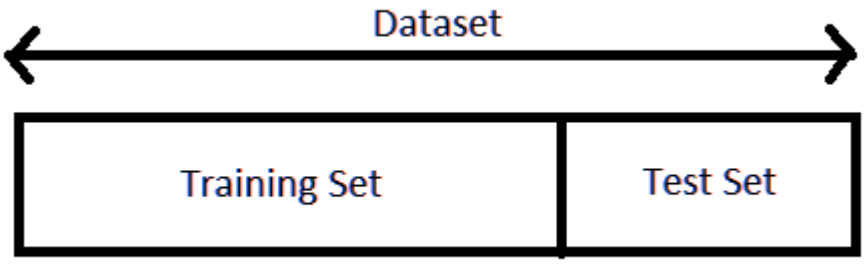


Ilustración 5 Del dataset extraemos el training y el test set

En nuestro proyecto Python hemos utilizado una de las múltiples herramientas que incluye la librería *Scikit-learn para realizar esta división*:

```
# Importamos de Scikit-learn utilidad para dividir los datos entre training y testing set
from sklearn.model_selection import train_test_split

# Dividimos los datos entre training (80 por ciento) y testing set (20 por ciento)
train_features, test_features, train_labels, test_labels = train_test_split(features, labels, test_size = 0.20, random_state = 51)
```

Con los datos del *training set*, lo que hacemos es dejar a nuestro algoritmo "ver" las soluciones, para que mediante ese entrenamiento pueda aprender a identificar los distintos ataques, y de esa forma cuando se enfrente a los datos del test set, pueda demostrar hasta qué nivel es certero en sus predicciones.

[27] (Gutierrez, 2015)

Entrenamiento de nuestro algoritmo

Con la librería Scikit-learn de Python resulta extremadamente simple crear y entrenar nuestro algoritmo de *random forest*. Tan solo necesitamos importar el constructor de RandomForestRegressor, definir el número de árboles que queremos que constituyan nuestro bosque, y entrenarlo con los datos que hemos reservado previamente para nuestro *training set*.

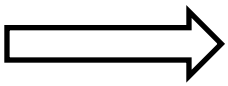
```
#Importamos la librería de Scikit-learn
from sklearn.ensemble import RandomForestRegressor

#Creamos un random forest compuesto por 200 árboles de decisión
randomForest = RandomForestRegressor(n_estimators = 200, random_state = 51)
```

En este punto nos hemos encontrado con un problema, y es que el clasificador no acepta variables no numéricas, por lo que hemos necesitado reemplazar esos valores substituyéndolos por valores numéricos.

Para evitar que nuestro algoritmo otorgue mayor peso a un valor que a otro porque éste tenga un valor numérico mayor, hemos hecho lo que se llama *one-hot encoding*, esto es convertir en columnas cada uno de los posibles valores y asignarle 1 cuando éste tiene lugar. Para el caso de la columna protocolo sería la siguiente conversión:

Protocolo
tcp
udp
icmp



tcp	udp	icmp
1	0	0
0	1	0
0	0	1

Esto lo hemos hecho mediante una función de la librería *pandas* llamada *get_dummies*:

```
features = pd.get_dummies(features)
```

A continuación hemos podido llamar a la función de entrenamiento sobre nuestro objeto randomForestRegressor:

```
#Finalmente entrenamos nuestro modelo con el training set
randomForest.fit(train_features, train_labels);
```

El pseudocódigo de lo que sucede cuando entrenamos un *random forest* sería el siguiente[28]:

INPUT
X={x_1,x_2,…x_n} ←muestras del training set
Y={y_1,y_2,…y_n} ←etiquetas reales correspondientes
ntree ←número de árboles de decisión
m_{try} ←número de propiedades de división
extra_options ←parámetro opcional
OUTPUT
model ←el random forest construido
Begin Procedure
IF 'extra_options' exist
    var = extra_options.var
ENDIF
IF ntree is null or netree <=0
    ntree = 200
ELSE ntree maintain original value
ENDIF
IF m_{try} exist or m_{try} <=0 or m_{try}>m
    m_{try} =√m
ELSE m_{try} maintain original value
ENDIF
    Run mexClassRF_train
RETURN model
Clear mexClassRF_train

[28] (Hu, 2014, pág. 36)

Ejecución de nuestro algoritmo

A continuación ejecutamos nuestro algoritmo para determinar si es capaz de identificar ataques de forma satisfactoria, partiendo de los datos del tráfico de red. Ese es el motivo para el que lo hemos entrenado mediante los datos del training set, que eran el 80 % de los datos que componen el dataset.

Debido a que nuestro equipo tenía escasa capacidad de procesamiento y no disponía de una GPU moderna, las pruebas de ejecución han sido muy lentas (alrededor de 20minutos de ejecución cada vez con el dataset del 10% del total), y nos ha sido imposible ejecutarlo con el dataset completo.

La ejecución del algoritmo es muy sencilla, basta con llamar la función *predict* sobre el objeto *RandomForestRegressor* que hemos creado previamente.

```
print(classification_report(x, test_labels))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	98755
1	0.94	1.00	0.97	50
micro avg	1.00	1.00	1.00	98805
macro avg	0.97	1.00	0.99	98805
weighted avg	1.00	1.00	1.00	98805

Hemos obtenido una precisión del 94 por ciento en la detección de ataques, y del 100 por ciento en la no detección de no ataques. Es decir, que nuestro algoritmo arroja algunos falsos negativos, pero no produce falsos positivos con el dataset que hemos seleccionado para este TFM.

Posiblemente este 94 por ciento mejoraría si pudiésemos utilizar el dataset completo con un equipo más potente.

Posible riesgo detectado

A pesar de que como hemos podido observar en el apartado anterior, un dataset de calidad nos permite entrenar nuestro algoritmo para que sea capaz de identificar de forma eficiente ciertos ataques de red, hemos identificado un riesgo inherente a esta estrategia de defensa.

En este TFM hemos partido de la base que disponemos de un dataset para entrenar nuestro algoritmo, pero en la práctica sería necesario entrenar nuestro sistema mediante datos reales y refrescarlo de forma regular mediante *logs* de nuestros servidores y dispositivos de seguridad perimetral, para mantenerlo actualizado y permitirle entrenar frente a nuevas amenazas. Esto por un lado resultará en un incremento de la robustez de nuestro algoritmo, pero por otro abrirá la puerta a nuevos riesgos como los ataques de envenenamiento[29], que mediante generación de gran volumen de datos malintencionados en los logs que nutren nuestro algoritmo, pueden lograr alterar el comportamiento global de éste. En la siguiente ilustración podemos ver el funcionamiento de esta estrategia de ataque.

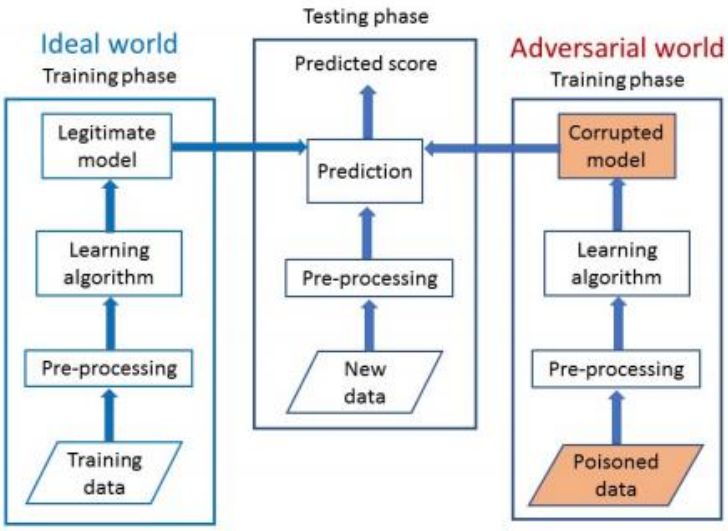


Ilustración 6 Envenenamiento de logs

Este tipo de ataques es posible que adquiera gran relevancia en los próximos años, dada la más que probable proliferación de tecnologías basadas en ML aplicadas a la seguridad. Debido al alcance limitado de este TFM no se ha podido indagar más, pero podría ser un tema de interés para el desarrollo de futuros trabajos de investigación.

[29] (Jagielski, y otros, 2018)

Conclusiones

Mediante la realización de este trabajo hemos tenido la oportunidad de ver cómo funciona un algoritmo ML basado en *random forest* para la detección de ataques DoS y DDoS, y hemos podido apreciar que disponer de un dataset amplio y de calidad es fundamental para obtener buenos resultados por parte del algoritmo.

También hemos podido apreciar que Python junto a sus librerías resulta una poderosa herramienta para trabajar con ML, aportando entre sus ventajas una gran compatibilidad con múltiples sistemas operativos, y una curva de aprendizaje mucho más corta que otros lenguajes específicamente orientados para análisis estadístico como sería por ejemplo el caso de R.

Este TFM nos ha brindado la oportunidad de ver a nivel práctico como funciona un algoritmo de *random forest* y ver una de las múltiples formas en que puede aplicarse al ámbito de la seguridad informática.

Por otra parte, nos ha permitido tener una primera idea de uno de los posibles riesgos que puede entrañar confiar toda nuestra seguridad a una estrategia basada en ML, como serían los ataques por envenenamiento.

Código fuente utilizado

```
#Importamos Pandas para manipulación de datos
import pandas as pd

#Leemos el dataset de la ruta donde lo tengamos
features = pd.read_csv('C:/tfm/kddcup.data.corrected.csv')


# One-hot encode con pandas get_dummies
features = pd.get_dummies(features)

# Usamos numpy para convertir a arrays
import numpy as np

# Labels son los valores que queremos predecir
labels = np.array(features['attack._nmap.'])
np.append(labels, [features['attack._normal.'],features['attack._perl.']])
np.append(labels, [features['attack._phf.'],features['attack._pod.']])
np.append(labels, [features['attack._portsweep.'],features['attack._satan.']])
np.append(labels, [features['attack._smurf.'],features['attack._teardrop.']])
np.append(labels, [features['attack._warezmaster.']])
labels = np

# Eliminamos los labels de los datos origen
# axis 1 hace referencia a las columnas
features= features.drop('attack._nmap.', axis = 1)
features= features.drop('attack._normal.', axis = 1)
features= features.drop('attack._perl.', axis = 1)
features= features.drop('attack._phf.', axis = 1)
features= features.drop('attack._pod.', axis = 1)
features= features.drop('attack._portsweep.', axis = 1)
features= features.drop('attack._satan.', axis = 1)
features= features.drop('attack._smurf.', axis = 1)
features= features.drop('attack._teardrop.', axis = 1)
features= features.drop('attack._warezmaster.', axis = 1)

# Guardamos los nombres de los parámetros para más adelante
feature_list = list(features.columns)

# Convertimos a numpy array
features = np.array(features)

# Importamos de Scikit-learn utilidad para dividir los datos entre training y testing set
from sklearn.model_selection import train_test_split
```

```
# Dividimos los datos entre training(80 por ciento) y testing set (20 por ciento)
train_features, test_features, train_labels, test_labels = train_test_split(np.array(features),
labels, test_size = 0.20)
# Importamos el modelo que estamos usando
from sklearn.ensemble import RandomForestRegressor

# Creamos un modelo con 200 árboles de decisión
rf = RandomForestRegressor(n_estimators = 200)

# Entrenamos el modelo con el training data
rf.fit(train_features, train_labels);

predicted= rf.predict(test_features)

#normalizamos a int
x = predicted.astype(int)

# Mostramos los resultados
from sklearn.metrics import classification_report
print(classification_report(x, test_labels))
```

Bibliografía

Adaptive Signal Control Technology. (s.f.). Obtenido de https://www.fhwa.dot.gov/innovation/everydaycounts/edc-1/asct.cfm

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning.* New York: Springer.

Alzahrani, S., & Hong, L. (2018). *Generation of DDoS Attack Dataset for Effective IDS Development and Evaluation.* Obtenido de https://file.scirp.org/pdf/JIS_2018081515001988.pdf

Benítez, R., Escudero, G., Kanaan, S., & Masip Rodó, D. (2014). *Inteligencia Artificial Avanzada.* Barcelona: Editorial UOC.

Bonaccorso, G. (2018). *Machine Learning Algorithms.* Bimingham: Packt Publishing.

Christmann, I. S. (2008). *Support Vector Machines.* New York: Springer.

Cole, M. (2018). *Hands-On Neural Network Programming with C#.* Birmingham: Paclt Publishing.

Faulconer, M. (2018). *Councilmember Zapf Announce City's Smart Traffic Signals Are Shrinking Commute Times*. Obtenido de https://www.sandiego.gov/mayor/news/releases/mayor-faulconer-councilmember-zapf-announce-city%E2%80%99s-smart-traffic-signals-are-shrinking

Graña, M., Toro, C., Posada, J., Howlett, R., & Lakhmi, C. (2012). *Advances in Knowledge-Based And Intelligent Information And Engineering Systems.* Amsterdam: The authors and IOS Press.

Gutierrez, D. (2015). *Machine Learning and Data Science. An Introduction to Statistical Learning Methods With R.* Basking Ridge: Technics Publications.

Hu, A. (2014). *Computer Science and Applications.* Wuhan: CRC Press.

Jagielski, M., Oprea, A., Biggio, B., Liu, C., Nita-Rotaru, C., & Li, B. (20 de 12 de 2018). *Manipulating Machine Learning: Poisoning Attacks and Countermeasures for Regression Learning.* Obtenido de arxiv.org: https://arxiv.org/pdf/1804.00308.pdf

Jin, H., Tino, P., Corchado, E., Byrne, W., & Yao, X. (2007). *Intelligent Data Engineering and Automated Learning.* Birmingham: Springer.

Kamila, N. (2017). *Advancing Cloud Database Systems and Capacity Planning With Dynamic Applications.* Hershey: IGI Global.

Kumar G., D. (2019). *Machine Learning Techniques for Improved Business Analytics.* Hershey: IGI Global.

Mason, A., & Newcomb, M. (2001). *Cisco Secure Internet Security Solutions.* Indianapolis: Cisco Press.

McAfee. (2017). *Overcome the Attacker Advantage with McAfee Endpoint Security.* Obtenido de https://www.mcafee.com/enterprise/en-us/assets/solution-briefs/sb-overcome-attacker-advantage.pdf

Merchán Macías, F. (s.f.). *openacccess.uoc.edu.* Obtenido de http://openaccess.uoc.edu/webapps/o2/bitstream/10609/81110/1/jmerchanmTFM0618.pdf

Miller, J. D. (2017). *Statistics for Data Science.* Birmingham: Packt Publishing.

Olleros, F., & Zhegu, M. (2016). *Research Handbook on Digital Transformations.* Glos: Edward Elgar Publishing.

Özgür, A., & Erdem, H. (2016). *A Review of KDD99 Dataset Usage in Intrusion Detection and Machine Learning between 2010 and 2015.* Obtenido de https://www.researchgate.net/publication/309038723_A_review_of_KDD99_dataset_usage_in_intrusion_detection_and_machine_learning_between_2010_and_2015

Singh Chauhan, A. (2018). *Practical Network Scanning.* Birmingham: Packt Publishing Ltd.

Sophos. (2018, Diciembre 13). *Intercept X Deep Learning.* Retrieved from https://www.sophos.com/en-us/medialibrary/PDFs/factsheets/sophos-intercept-x-deep-learning-dsna.pdf

Sud, R., & Edelman, K. (2004). *SECUR Exam Cram 2.* United States of America: Que Publishing.

Swanson, L. W. (2017). *The beautiful brain: The Drawings of Santiago Ramon y Cajal.* New York: Abrams.

Vision Zero Labs: Using Data Science to Improve Traffic Safety. (s.f.). Obtenido de https://blogs.microsoft.com/newyork/2017/06/29/vision-zero-labs-using-data-science-to-improve-traffic-safety/

Yan, X. (2009). *Linear Regression Analysis.* Singapore: World Scientific Publishing.