Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO DE FIN DE GRADO

Grado en Ingeniería Informática
Mención en Tecnologías de la Información

Sistemas de Detección de Intrusos Basados en Técnicas de Machine Learning

Autor: D. Gonzalo Valdezate Álvarez

Tutor: D. Valentín Cardeñoso Payo

Índice general

Resumen

El número de dispositivos que utilizan Internet, así como las tareas que se realizan a través de él, aumenta cada día. También lo hacen los ataques contra la disponibilidad, integridad y confidencialidad de la información que estos manejan. Los Sistemas de Detección de Intrusos (IDS) son uno de los mecanismos de seguridad más efectivos para proteger sistemas en red contra ataques informáticos, se tenga o no conocimiento previo de ellos. Destaca la aplicación de la Inteligencia Artificial y, más concretamente, del aprendizaje automático en este tipo de programas. La mayor parte de estos IDS analizan el tráfico de la red y su comportamiento normal de forma que pueden activar una alarma cuando parte de dicho tráfico difiere de lo habitual. De esta forma se puede detectar algunos tipos de ataque aunque nunca se hayan realizado antes. En este documento analizaremos qué algoritmos de aprendizaje automático son más adecuados para detectar determinados tipos de ataques basándonos en este principio.

Palabras clave:
Sistemas de Detección de Intrusos, IDS, Machine Learning, Ciberseguridad.

Capítulo 1

Introducción

En la actualidad el uso de Internet se ha convertido en algo común en todos los aspectos de la vida. La inmensa mayoría de procesos cuya información se almacenaba antes en papel se han modificado para realizarse a través de la red y utilizar únicamente el formato digital. Datos personales de gran importancia (tales como números de cuenta, salarios, informes médicos o datos biométricos) se transmiten y almacenan en los equipos conectados a Internet. Es por ello que los ataques informáticos contra los equipos que soportan estas operaciones y almacenan estos datos han ido creciendo exponencialmente. También ha evolucionado la forma en la que nos protegemos de ellos. No obstante, estos ataques cambian y se adaptan constantemente a las nuevas tecnologías. Una de las mejores formas de proteger una red, ya sea frente a ataques conocidos como frente a aquellos de los que aún no se tiene constancia, son los Sistemas de Detección de Intrusos (*IDS*, por sus siglas en inglés).

Los Sistemas de Detección de Intrusos son programas informáticos utilizados para detectar accesos no autorizados o maliciosos a un ordenador o una red. Esto se realiza generalmente mediante el análisis de los paquetes recibidos por una determinada interfaz de red. Cuando el IDS detecta un paquete o conjunto de paquetes que puedan corresponderse a un ataque informático, genera una alarma, que típicamente tendrá que ser atendida por un técnico en seguridad. Lo deseable sería que el programa reaccionase únicamente ante paquetes que realmente se corresponden a un ataque, pero esto es imposible. El IDS nunca sabrá a ciencia cierta si un paquete corresponde a un ataque o no; deberá utilizar una serie de técnicas para tratar de determinarlo. En función del tipo de sistema, será deseable una mayor o menor sensibilidad frente a paquetes sospechosos.

El objetivo de este documento es determinar en qué medida los algoritmos de machine learning pueden aplicarse a la detección de intrusiones y qué algoritmos son más apropiados para detectar qué tipos de ataques en función de unas determinadas métricas. Para ello se realizará un estudio práctico aplicando una serie de algoritmos de machine learning a un dataset que contenga la información correspondiente a un conjunto de paquetes de red. Parte de los mismos corresponderán al uso legítimo de la red y parte a ataques informáticos. Los paquetes correspondientes a ataques estarán marcados de antemano para poder evaluar la eficacia de cada algoritmo. Así, se pretende determinar qué algoritmos son más adecuados en función de qué métricas se consideren más importantes (Tasa de falsos positivos, tasa de falsos negativos, precisión...) y de qué tipo de ataques sea prioritario detectar. El dataset será seleccionado de entre los disponibles en Internet para esta

finalidad.

Capítulo 2

Ataques contra sistemas en red

Consideramos un **ataque contra un sistema en red** cualquier acción que tenga como objetivo comprometer la integridad, la confidencialidad o la disponibilidad de sus datos. Es decir, cualquier acción que comprometa la seguridad de la información de un sistema. Así, la propia definición de qué constituye o no un ataque contra una red de ordenadores depende de qué red se trate, de su función y de su política de seguridad. Esto dificulta la detección de estos ataques, ya que un mismo evento puede considerarse o no un ataque dependiendo de en qué sistema se produzca. No obstante, teóricamente, si se dispusiese de un modelo perfecto para diferenciar el tráfico legítimo en una red del malicioso se podría detectar cualquier intrusión. En la práctica esto es imposible en cualquier situación mínimamente compleja. Por tanto, la cuestión está en si es posible obtener un modelo aproximado lo suficientemente bueno como para ofrecer un rendimiento adecuado.

Como hemos dicho los ataques informáticos cambian constantemente y evolucionan para tratar de evitar los mecanismos de seguridad que se implementan para evitarlo. Aún así, es necesario tener ciertos conocimientos sobre los principales tipos de ataques conocidos y sus categorías, para poder prevenirlos y tratar de mitigarlos adecuadamente en cada caso.

2.1. Clasificación de los ataques

Existen numerosos tipos de ataque que pueden afectar a los sistemas en red y existen muy diversas formas de clasificarlos.

En esta sección no se pretende hacer un estudio exhaustivo de todos, tan solo se pretende presentar las principales familias de ciberataques con el fin de hacer referencia a ellas en el estudio posterior; y así poder determinar la eficacia de un IDS para detectar cada una de ellas. Para hacerlo nos basaremos en la misma clasificación que suele realizarse a la hora de estudiar la efectividad de sistemas de detección de intrusiones, desde la presentación en 1998 del dataset DARPA para este fin [1]. Esta clasificación considera que todos los ataques contra sistemas en red pueden abarcarse con 4 grandes categorías: Probing, Remote to Local, User to Root y Denial of Service. Esta división resulta útil ya que separa los ataques en varias categorías en las que el comportamiento del atacante en la red es muy diferente, motivo por el cual se utiliza a la hora de aplicar algoritmos de machine learning para la detección y clasificación de ataques: facilita la creación de un modelo por parte del algoritmo.

Más tarde haremos referencia a las familias de ataques consideradas por esta clasificación para evaluar la eficacia de los IDS.

2.1.1. Ataques en función del objetivo

- **Probing/Scanning**. Ataques que tratan de escanear una red o una máquina en busca de información y de posibles vulnerabilidades. En muchas ocasiones se trata de procesos automatizados realizados por herramientas como **Nmap** o **Zenmap**. Uno de los más habituales es el **escaneo de puertos**. Consiste en la realización de peticiones contra diferentes puertos de las máquinas de una red para comprobar si están abiertos, si ofrecen algún servicio y, en ese caso, si esto puede suponer alguna vulnerabilidad. Una vez conocidos los servicios prestados por una máquina lo habitual es buscar estas vulnerabilidades examinado la versión del software asociado a ellos y realizando peticiones maliciosas para obtener más información. Por ejemplo, lo habitual cuando se quiere atacar contra servidores web es comprobar si son susceptibles a SQLi o a XSS, de los que hablaremos más adelante. Es decir, los ataques pertenecientes a esta familia solo buscan obtener información, generalmente con el fin de poder aplicar algún otro tipo de ataque contra algún recurso de la red objetivo. Aunque en gran parte de los casos solo acceden a información cuya consulta es legitima, siguen considerándose ataques por ser este su objetivo final, de acuerdo a la definición dada anteriormente. Pueden considerarse como la primera fase de cualquier tipo de ataque informático.

- **User to Root (U2R)**. Ataques en los que el atacante ya posee acceso a un recurso en un sistema y trata de explotar alguna vulnerabilidad a nivel de aplicación o de sistema operativo para obtener privilegios más elevados. Uno de los ejemplos más clásicos son los ataques de desbordamiento de buffer (**buffer overflow**), en los que el atacante introduce valores específicos en la entrada de aplicaciones vulnerables de forma que los datos introducidos desborden la pila del programa y se escriban sobre la sección de memoria correspondiente al código, de forma que pueda ejecutarse código arbitrario con los privilegios propios de la aplicación. Esto ocurre cuando el programa no controla adecuadamente la cantidad de datos escrita sobre un área de memoria reservada para tal fin (buffer). En general todos los ataques de inyección de código caen dentro de la categoría U2R. Dos tipos de ataque que se dan con gran frecuencia contra servidores web son la inyección SQL y el Cross Site Scripting, que ya hemos mencionado antes. La **inyección SQL** (SQLi) consiste en la realización de peticiones SQL maliciosas que, si el servidor no controla adecuadamente la entrada de usuario, pueden permitir al atacante leer información confidencial de la base de datos o modificarla. El **Cross Site Scripting** (XSS) consiste en la inserción de scripts maliciosos en servidores web legítimos, de forma que sean ejecutados cuando otro usuario acceda a ellos. Existen numerosos **exploits** que permiten obtener privilegios de administrador aprovechando estas u otras vulnerabilidades. Resulta también reseñable el comportamiento de las **rootkits**, un tipo de malware que posee la capacidad de ocultarse en el sistema operativo, ocultarse de sus usuarios y ofrecer al atacante acceso remoto con privilegios de administrador. Otros ejemplos de ataques U2R serían: ataques con PowerShell, Xterm, etc.

- **Remote to Local (R2L)**. Ataques en los que se intenta explotar vulnerabilidades

de un sistema de forma remota para tratar de obtener privilegios de un usuario local. Generalmente son precedidos de un escaneo de recursos de la red. Para acceder a ellos generalmente debe conseguirse autenticarse contra alguno de los servicios ofrecidos por las máquinas de la red, evitando cualquier mecanismo de seguridad que haya de por medio (Firewall, WAF, IPS...). Esto puede realizarse obteniendo las credenciales para estos servicios o aprovechando alguna vulnerabilidad en ellos que nos permita evitar (*bypass*) el mecanismo de autenticación. El método más básico para obtener unas credenciales es el **ataque de fuerza bruta**: se van realizando peticiones con diferentes credenciales hasta que alguna autenticación sea exitosa. Otros métodos para obtener una contraseña serían la ingeniería social y, más concretamente, el **phishing**. Aunque un acceso utilizando estas credenciales no diferiría en nada de un acceso legítimo, sí que es posible detectar una campaña de phishing dirigida contra los usuarios de una red. Otra forma de detectar credenciales serían los ataques de tipo **man-in-the-middle**, en los que el atacante se sitúa en algún punto de la red donde pueda interceptar las comunicaciones entre dos hosts y utiliza esta información con fines maliciosos. Hoy en día esa información suele estar encriptada, pero el atacante sigue pudiendo intentar desencriptarla a la fuerza o replicar paquetes para autenticarse contra algún sistema. Los ataques de **buffer overflow** también son una forma habitual de explotar una aplicación de servidor (Apache, Imap, Senmail...) para obtener acceso al equipo.

- **Denial of Service (DoS)**. Ataques que buscar comprometer la disponibilidad de los datos de un sistema. Generalmente bloquean un recurso de una máquina o red saturándolo con peticiones ilegítimas de forma que sus usuarios legítimos no tengan acceso a él. Las peticiones maliciosas pueden intentar saturar el objetivo tanto por su número como por su contenido. Por ejemplo en los ataques de tipo **Teardrop** (ya obsoletos) el atacante enviaba múltiples paquetes TCP fragmentados para consumir los recursos de la máquina objetivo cuando ésta intente reensamblarlos. Los fragmentos se enviaban de tal forma que el SO de la máquina objetivo no pudiese ensamblarlos. Otro ejemplo serían los ataques de tipo **UDP Flood** (o UDP storm) en los que el atacante colapsa puertos arbitrarios de la máquina destino con paquetes UDP. Este tipo de ataque sigue produciéndose en la actualidad. La máquina atacada busca aplicaciones asociadas a esos puertos y, al no encontrar ninguna en la mayoría de los casos, devuelve un paquete indicando que el destino es inalcanzable. En este caso se colapsa el destino basándose solamente en la cantidad de paquetes enviados, mientras que Teardrop se basaba también en su contenido. Una forma común de realizar ataques de denegación de servicio es mediante una **botnet** (un conjunto de ordenadores infectados) utilizada para realizar estas peticiones (**Distributed Denial of Services, DDoS**). Con el fin de atravesar algunos de los mecanismos de defensa de una red es habitual que los atacantes falseen las direcciones IP de origen (**IP Spoofing**).

Existen muchas otras clasificaciones más exhaustivas para los ataques informáticos. Desde el punto de vista de la seguridad de red, una de las más utilizadas es la propuesta por The Mitre Corporation, que clasifica las diferentes técnicas utilizadas por los atacantes en 12 grupos en función de sus objetivos [3]. Esta clasificación es pública y actualmente es utilizada por empresas y organizaciones de todo el mundo para evaluar su grado de protección frente a los diferentes tipos de ataque informático conocido. El lector que quiera conocer una clasificación más precisa de estos ataques, así como las técnicas utilizadas y

ejemplos de ellas, puede consultarla en su página web. Para el estudio realizado en este documento nos centraremos en la clasificación anteriormente expuesta, ya que nuestro objetivo es evaluar la eficacia del machine learning para separar el tráfico legítimo del malicioso y hacer una clasificación de este último. Esto será mucho más sencillo si clasificamos todos los ataques en uno de esos cinco grande grupos.

2.1.2. Ataques informáticos y anomalías en la red. Ataques pasivos y ataques activos

Para el tema que nos ocupa en este documento, nos interesa conocer hasta qué punto los diferentes ataques los ataques informáticos provocan eventos detectables en una red. Existe otra posible clasificación de los ataques que resulta útil desde ese punto de vista:

- **Ataques pasivos**. No afectan a los recursos del sistema. Simplemente intentan hacer uso de ellos o conocerlos sin modificarlos. Algunos de estos ataques son imposibles de detectar, ya que no generan ningún elemento detectable (por ejemplo, un paquete de red) accesible desde la red atacada. Algunos ejemplos serías el network sniffing, los escaneos silenciosos o los ataques de tipo man-in-the-middle. Estos ataques serían imposibles de detectar mediante un IDS.

- **Ataques activos**. Realizan alguna modificación no autorizada de los elementos de la red o de su flujo de datos, de forma que siempre generan algún elemento detectable. La gran mayoría de los ataques en red pertenecen a esta categoría. El problema en este caso es, como es habitual, poder diferenciarlos del uso legítimo de la red.

Por tanto hay que tener en cuenta en todo momento que no todos los ataques son detectables y que un IDS siempre deberá formar parte de una infraestructura de seguridad de red más amplia, que incluya otros mecanismos de seguridad que permitan prevenir otros tipos de ataque.

Capítulo 3

Sistemas de Detección de Intrusos

Como se ha dicho, los IDS son mecanismos de seguridad software cuya funcionalidad consiste en detectar accesos ilegítimos a una red o equipo. Al hacerlo, el IDS generará una alerta de seguridad que típicamente tendrá que ser revisada por un técnico que determinará si se trataba de un falso positivo o si realmente ha habido un ataque y, en este último caso, si el ataque ha tenido o no impacto en la red. Si lo hubiese tenido, tendrían que tomarse también las medidas necesarias para revertir o mitigar ese impacto. La detección de intrusos se basa en la premisa de que el comportamiento de los intrusos difiere del de un usuario legítimo [4]. Existen diferentes tipos de IDS en función de qué datos analizan y de qué tipo de técnicas utilizan para detectar posibles ataques.

Los IDS pueden monitorizar información de toda una red o únicamente de un equipo. Aquellos que monitorizan únicamente el dispositivo en el que están instalados se denominan **Sistemas de Detección de Intrusos basados en Host** (HIDS, por sus siglas en inglés). Los HIDS pueden monitorizar, además de los paquetes que llegan a una determinada interfaz de red, elementos muy diferentes, como archivos de log o procesos en ejecución, por lo que son bastante dependientes de la veracidad de esta información [5]. Por otro lado, los **IDS basados en Red** (NIDS) analizan información relativa a toda una red. Generalmente son instalados en diferentes equipos situados en determinados puntos de una misma red informática para tener una visión global de lo que ocurre en la red. Estos sistemas pueden analizar dicho tráfico utilizando muy diversas técnicas para determinar si su propósito es ilegítimo. Tienen el inconveniente de que, obviamente, solo pueden detectar aquellos ataques que utilizan la red y solo mientras la están utilizando. Aunque tradicionalmente este tipo de IDS analizaba el tráfico de red visible desde sus sondas, hoy en día es común que analicen también ficheros de log llegados desde los diferentes puntos de la red, especialmente desde elementos estratégicos como el Firewall. En cuanto al análisis del tráfico de red, un NIDS puede analizar directamente los paquetes capturados en una o varias interfaces de red o analizar estructuras de datos de más alto nivel extraídas de este tráfico. Generalmente estas estructuras se basan en la recolección de datos correspondientes a un mismo flujo de paquetes, entendiéndose un **flujo** como un conjunto de paquetes con iguales direcciones IP y puertos de origen y destino, e igual protocolo de la capa de transporte. En este documento nos centraremos en el estudio de los NIDS, por ser los más utilizados y los que mejores resultados proporcionan [6].

De manera complementaria a la clasificación anterior, los IDS pueden clasificarse en función de las técnicas que utilizar para determinar si un paquete o conjunto de paquetes

corresponde a un ataque.

- Los **IDS basados en firmas** analizan el contenido de los paquetes y lo comparan con una base de datos de firmas conocidas correspondientes a ataques ya documentados. Un paquete o conjunto de paquetes se descompone en una serie de características que en conjunto determinan una firma. Si la firma de parte del tráfico de red se corresponde con alguna de las de la base de datos, éste se marcará como malicioso [5]. El administrador también puede crear sus propias firmas para que salte una alarma cuando ciertas situaciones se produzcan en su red. Este tipo de IDS no sirve para detectar ataques de los que no se tenga constancia previa; aunque, si se utilizan las firmas adecuadas para cada red, puede tenerse una seguridad bastante alta de que, cuando genera una alarma, se ha detectado realmente un ataque.

- Los **IDS basados en anomalías** tratan de modelar el tráfico legítimo y habitual de una red analizándolo. Cuando un paquete o flujo de paquetes difiere de este modelo se marca como malicioso y se genera una alerta. De esta forma este tipo de IDS se ajusta a la idea expuesta anteriormente de que la clasificación de un evento como ataque informático depende del sistema en el que ocurra y de su política de seguridad. De esta forma un IDS basado en anomalías sí que puede detectar ataques que nunca se hayan visto antes, aunque puede generar alarmas para eventos legítimos o ignorar aquellos que no lo son. Todo depende cuanto se aproxime a la realidad del modelo utilizado para diferenciar los eventos maliciosos de los legítimos. Puesto que es necesario crear dicho modelo del comportamiento del sistema, estos programas no pueden instalarse y funcionar directamente en uno, deben de proporcionarse mecanismos para la construcción del mismo. Es aquí donde juegan un papel importante las técnicas de machine learning, puesto que proporcionan una herramienta para que el IDS aprenda el comportamiento normal de un equipo o una red e identifique cualquier anomalía que se salga de éste.

Teniendo en cuenta estos conceptos podemos concretar más el propósito del estudio realizado en este documento. Este será únicamente determinar qué algoritmos de machine learning son más adecuados, en un NIDS basado en anomalías, a la hora de construir un modelo del tráfico de red y de detectar anomalías en el mismo. En estos casos la detección de ataques suele considerarse como un problema de clasificación [6], ya que su objetivo es clasificar cada paquete o flujo de paquetes como normal o malicioso.

En la siguiente sección veremos los principios básicos del machine learning y su aplicación en tareas de clasificación. Pero antes, vamos a repasar algunas de las soluciones IDS más populares actualmente, fijándonos en si implementan o no la detección de anomalías y en cómo lo hacen.

3.1. Principales soluciones IDS existentes

Actualmente existen muchas soluciones IDS en el mercado, tanto de código abierto como de pago. Algunas están pensadas para redes pequeñas e incluso redes domésticas, mientras que otras se aplican a servicios cloud con gran cantidad de tráfico muy variado. Estas son algunas de las más conocidas:

- OSSEC

- Snort
- Suricata
- Zeek/Bro

La mayoría de estas soluciones ni siquiera ofrecen la posibilidad de la detección de anomalías. La única que lo hace es Zeek, de forma conjunta con la detección basada en firmas. Es por esto que consideramos que el futuro cercano de la detección de intrusiones basada en anomalías se encuentra en su uso de forma conjunta con la detección basada en firmas. Esta última ya puede generar una gran cantidad de falsos positivos si no se calibra adecuadamente, de forma que la aplicación de técnicas basadas en detección de anomalías solo se realizará en la práctica cuando pueda conseguirse un grado de fiabilidad igual o mayor a estas, y se consiga recopilar suficiente información como para que un analista pueda investigar fácilmente si realmente ha habido un ataque y qué impacto ha tenido. El mero hecho de detectar tráfico anómalo no es suficiente para detectar y tratar intrusiones en un sistema real.

Capítulo 4

Machine Learning

Llamamos **Machine Learning** (ML o Aprendizaje Automático) a la rama de la Inteligencia Artificial cuyo objetivo es desarrollar técnicas que permitan que un programa informático aprenda. De esta forma, un algoritmo de machine learning es un algoritmo que es capaz de **aprender** a partir de unos datos. "Consideramos que un programa aprende de una experiencia E respecto a una tarea específica T y a una medida de rendimiento R; cuando su rendimiento en la tarea T, de acuerdo a la métrica R, aumenta con la experiencia E" [7].

Dada la definición anterior, el principal problema consiste en determinar a qué posibles tareas, experiencias y métricas puede aplicarse el machine learning. A continuación trataremos de dar una visión general de los tipos de problemas a los que se pueden aplicar los algoritmos de ML actuales, las métricas que se pueden utilizar para medir su desempeño en ellas y del tipo de experiencia mediante la que los algoritmos aprenden. Haremos esto centrándonos en los principios expuestos en [7] y [8]. Una vez esbozadas estas ideas, nos centraremos en la aplicación que nos ocupa, la detección de intrusiones en una red.

4.1. Aplicaciones de los algoritmos de machine learning

El machine learning nos permite afrontar tareas que son demasiado complejas como para resolverlas con programas tradicionales escritos y diseñados completamente por seres humanos. Es importante destacar que la tarea que realiza el algoritmo es diferente del proceso de aprendizaje. Si queremos un software que traduzca entre dos idiomas diferentes podemos tratar de programar manualmente la forma en que debe hacerlo o programar un software que tenga la capacidad de aprender a traducir. En ambos casos la tareas es la misma.

A continuación se listan algunos de los tipos de tareas más habituales. De nuevo, esto no es una definición exhaustiva de un número fijo de tareas que los algoritmos de ML pueden realizar. Tan solo son algunos de los campos en los tradicionalmente se divide el tipo de actividad al que se aplican. Un mismo algoritmo de ML puede usarse para varias ellas [8].

- Regresión: Predecir del valor de una variable o vector $\mathbf{y}$ a partir de un vector $\mathbf{x}$.

- Clasificación: Clasificar cada elemento de un conjunto de datos como una entre N etiquetas
- Agrupamiento o clustering: Agrupar los elementos de un conjunto de datos en subconjuntos similares.
- Output estructurada: Las tareas de este tipo incluyen cualquier actividad en la que la salida es un vector estructurado, es decir, en el que no solo sus campos contienen información, sino que también lo hace la forma en la que esos campos están relacionados. Esto incluye problemas de traducción de lenguaje, reconocimiento de escritura manual, etc.
- Reducción de la dimensionalidad: Reducción del número de variables que componen los datos de entrada de forma que queden solo las más relevantes.
- Estimación de densidades: Aproximar la función de densidad de probabilidad o la función de masa de probabilidad de un conjunto de datos
- Denoising, síntesis, sampling...

Como hemos dicho, esta lista no incluye todas las posibles tareas de machine learning, ni todas las tareas tienen que corresponderse a una sola categoría. Por ejemplo, cualquier problema de clasificación puede considerarse como uno de regresión, ya que el objetivo es determinar la probabilidad de que un elemento $\mathbf{x}$ de un conjunto de datos pertenezca a una categoría y (predecir y a partir de $\mathbf{x}$). Así, existen diferentes enfoques que pueden aplicarse a la hora de abordar un mismo problema utilizando machine learning.

4.2. Métricas de rendimiento en algoritmos de machine learning

Existen muchas formas de evaluar el rendimiento de un algoritmo de machine learning. Dependen en gran medida de el tipo de tarea que se trata de realizar. Nos centraremos en las métricas utilizadas para la detección de intrusiones en la sección 6.2.

Además de evaluarse el rendimiento del algoritmo a la hora de realizar su tarea, suele resultar importante a la hora de comparar algoritmos el consumo que hacen de los recursos del sistema (tiempo, memoria, CPU...).

4.3. Proceso de aprendizaje en algoritmos de machine learning

Los algoritmos de machine learning aprenden procesando **datos**. Los datos que procese un determinado algoritmo para aprender serán ejemplos del tipo de datos que procesará en el momento en el que realice la tarea para la cual se está aplicando. Estos datos serán un conjunto de características cuantitativamente medidas de un objeto o evento que queramos que el sistema aprenda a procesar. Un **dataset** es una colección de datos tal y como se acaban de definir [8].

Existen dos tipos de algoritmos de machine learning en función de la forma en la que procesan los datos durante su fase de aprendizaje.

- En el **aprendizaje no-supervisado** se procesa un dataset para obtener propiedades útiles de su conjunto de datos. De esta forma el sistema aprende qué esperar de los datos de entrada; construye un modelo aproximado de los valores de las diferentes características y las relaciones entre ellos.

- En el **aprendizaje supervisado** se procesa un dataset en el que cada dato tiene un conjunto de características siendo una su etiqueta u objetivo. Esta característica dependerá del sistema de aprendizaje en cuestión. Un sistema de reconocimiento de escritura podrá ser entrenado con un conjunto de imágenes asociadas a la letra que representan. Así, el sistema aprenderá viendo dibujos de la letra 'a' sabiendo que representan dicha letra. En un NIDS se mostrará al sistema eventos de la red y se le indicará a qué categoría pertenecen (eventos legítimos o maliciosos).

De forma aproximada, el aprendizaje no-supervisado implica observar diferentes ejemplos de un vector $\mathbf{x}$ y tratar de aprender la distribución de probabilidad $p(\mathbf{x})$ o algunas propiedades útiles de dicha distribución; mientras que el aprendizaje supervisado implica observar diferentes ejemplos de $\mathbf{x}$ asociados a un determinado valor o vector $\mathbf{y}$, para aprender a predecir $\mathbf{y}$ en función de $\mathbf{x}$, generalmente estimando la función de probabilidad de $\mathbf{y}$ dado $\mathbf{x}$, $p(\mathbf{y}\|\mathbf{x})$ [8].

Aprendizaje supervisado y no-supervisado no son términos definidos de forma estricta. Muchas tecnologías de machine learning pueden operar de ambos modos, lo que a menudo depende de la tarea que se pretende desempeñar. Además, existen otras variantes del paradigma de aprendizaje. En el aprendizaje semi-supervisado, algunos ejemplos vienen marcados con una etiqueta y otros no. En el aprendizaje por refuerzo (reinforcement learning) los algoritmos interactúan con un entorno, de modo que existe un bucle de retroalimentación entre el sistema y sus experiencias [8]. En este documento no nos centraremos en estos modelos.

4.4. Machine learning e IDS. Algoritmos de clasificación

Teniendo claros estos principios básicos, podemos volver a centrarnos en el objetivo de este documento. La detección de intrusiones en un NIDS mediante detección de anomalías suele considerarse un problema de clasificación, y en los últimos años se ha estudiado una serie de aproximaciones a ella muy diferentes [2] [1].

A la hora de detectar intrusiones puede tratarse de determinar una clasificación binaria de los eventos detectados, catalogándolos como maliciosos o legítimos. Puesto que típicamente los eventos catalogados como maliciosos harán saltar una alarma y necesitarán de la posterior revisión por parte de un técnico en seguridad, dicha distinción no es suficiente. Es necesario poder clasificar el evento como perteneciente a una familia de ataque informático siempre que sea posible, de acuerdo a lo expuesto en la sección 2, para que el técnico pueda revisar correctamente el incidente de seguridad y determinar si se ha producido un ataque realmente. Será necesario que se conozca al menos el tipo de ataque para que este sistema se aplique en la práctica; y se pueda determinar si el ataque ha tenido impacto, si se trataba de un falso positivo, etc. Sin esta característica la detección de anomalías no podría implantarse de forma efectiva en Sistemas de Detección

de Intrusos reales.

Además de técnicas de clasificación, también es bastante común el uso de técnicas de clustering [1] para la detección de intrusiones. Teniendo esto en cuenta hay diferentes aproximaciones que se pueden utilizar a la hora de aplicar los algoritmos de machine learning en un NIDS.

- **Clasificadores simples**. Puede utilizarse un único algoritmo de ML para tratar de agrupar los diferentes eventos analizados como legítimos o maliciosos y, en este último caso, para determinar el tipo de ataque.
- **Clasificadores híbridos**. Otra opción bastante utilizada en la bibliografía [2] es combinar varios algoritmos diferentes, aplicando uno al resultado del anterior. Una opción habitual es aplicar algún algoritmo de clustering antes de aplicar algún clasificador. También suelen aplicarse varios clasificadores (ya sean algoritmos diferentes o el mismo varias veces) para ir realizando clasificaciones sucesivas sobre cada tipo de ataque. De esta forma cada clasificador solo tiene que hacer una clasificación binara. Esta aproximación se muestra en [10].
- **Clasificadores compuestos** (ensemble classifiers). Esta técnica consiste en aplicar varios clasificadores sobre el mismo dataset y combinar los resultados de todos ellos. La forma más habitual de hacerlo es clasificar cada elemento por la categoría elegida por la mayoría de algoritmos.

De la misma forma, como ya se ha comentado, la detección de anomalías por parte de un IDS puede realizarse tanto de forma aislada como de forma conjunta con la detección basada en firmas.

Más adelante se describirán las técnicas concretas que se implementarán para el estudio.

Capítulo 5

Metodología

En esta sección se detallará más en profundidad la metodología que se utilizará para la realización del estudio práctico.

De esta forma, el primer paso será seleccionar las métricas que se tendrán en cuenta para determinar la idoneidad de un algoritmo de machine learning para la clasificación de un paquete o flujo de paquetes en las cinco categorías que nos interesan: Legítimo, Probe, U2R, R2L o DoS.

En segundo lugar, se seleccionará un dataset con una captura del tráfico de una determinada red, en la que cada paquete aparezca marcado con la categoría a la que corresponde, para poder aplicar técnicas de aprendizaje supervisado. Existen datasets disponibles en Internet que cumplen éstas características. Se estudiarán y se seleccionará uno en función del formato de sus datos, la cantidad de información que contenga, la variedad de los ataques que muestre y cómo de recientes sean sus datos. También se realizará un análisis y una presentación de las principales características relevantes del dataset, la distribución de los ataques respecto al número total de entradas, etc. A continuación se seleccionarán los atributos de dicho dataset que se desea tener en cuenta para la detección de ataques. Esta selección se hará de forma automática mediante ciertos algoritmos diseñados para ello. Se aprovechará para estudiar la eficacia de los algoritmos de clasificación en función de las características elegidas.

En tercer lugar, se seleccionará un conjunto de algoritmos de clasificación para ser considerados en el experimento. Se seleccionarán en función de su efectividad como algoritmos de clasificación generales y de su uso en otros estudios similares, tras una revisión de la bibliografía disponible. Muchos de estos algoritmos tendrán parámetros que será necesario determinar antes de aplicarlos a los datos.

Finalmente, se aplicará cada uno de los algoritmos a los datos del dataset. Se dividirá el dataset en dos partes. Una fracción mayoritaria del mismo se utilizará para entrenar al algoritmo de clasificación, es decir, para permitirle formarse un modelo del tráfico legítimo de la red a partir de las etiquetas de cada paquete. La fracción restante se utilizará para ver la eficacia de ese algoritmo a la hora de clasificar una vez creado el modelo. Para cada algoritmo se recogerán los resultados y se calcularán las métricas de rendimiento mencionadas anteriormente. También se tratará de calcular esas métricas para cada uno de los principales tipos de ataque marcados en el dataset. Se comparará esta información

entre los diferentes algoritmos y se tratará de extraer conclusiones sobre la eficacia de éstos en cada caso.

Capítulo 6

Estudio práctico

En esta sección se especificará el proceso de experimentación con los diferentes algoritmos de clasificación, documentando en detalle toda la información referente a la ejecución de los pasos descritos en la sección 5.

6.1. Tecnología a utilizar

Implementación de un script en **Python3** que, para cada uno de los algoritmos, aplique el algoritmo de ML al dataset. Primero para entrenar el modelo y posteriormente para probar su eficacia. Se estudiará introducir una fase de validación entre ambas. Este script se basará en las librerías Scikit-Learn y Plotly.

Scikit-Learn (también conocido como Sklearn), es una biblioteca de aprendizaje automático de software libre para Pyhton. Esta librería incluye los principales algoritmos de ML, así como herramientas para el tratamiento anterior y posterior de los datos. Estas herramientas incluyen utilidades para la división de los datasets y para la extracción y análisis de los datos resultantes de un test. Se selecciona Scikit-learn frente a otras tecnologías (Java WEKA, etc.) por su versatilidad y por la cantidad de documentación disponible.

Plotly es una librería de representación gráfica de datos que usaremos para representar los resultados de la prueba.

6.2. Selección de métricas

Existen diferentes métricas que se pueden considerar a la hora de determinar la idoneidad de un algoritmo de clasificación. Por un lado debemos considerar la **eficacia del algoritmo a la hora de clasificar** los elementos analizados. Para medirla utilizaremos las siguientes métricas:

- **Verdadero Positivo** (VP): representa el número de paquetes maliciosos clasificados correctamente como tales.
- **Verdadero Negativo** (TN): representa el número de paquetes legítimos clasificados correctamente como tales.
- **Falso Negativo** (FN): representa el número de paquetes maliciosos clasificados erróneamente como legítimos.

- **Falso Positivo** (FP): representa el número de paquetes legítimos clasificados erróneamente como maliciosos.
- **Exactitud** (Accuracy, ACC) [1]: Representa la fracción de los paquetes clasificados correctamente frente a los paquetes clasificados en total. También es conocida como precisión en ocasiones, no confundir con la precisión de la que hablaremos más adelante. Es una de las métricas de rendimiento más básicas en un algoritmo de clasificación. Se calcula mediante la siguiente fórmula:

$$ACC = (VP + VN)/(VP + VN + FP + FN) \tag{6.1}$$

- **Precisión** (P, precision o positive predictive value) [1]: Representa el porcentaje de positivos que realmente correspondían a un ataque. Una precisión alta significa pocos falsos positivos.

$$P = VP/(VP + FP) \tag{6.2}$$

- **Sensibilidad** (S, True Positive Rate o recall): Representa el porcentaje de ataques que han sido detectados correctamente. Un TPR alto significa pocos falsos negativos.

$$S = VP/(VP + FN) \tag{6.3}$$

- **Fscore**: Media armónica de la precisión y la sensibilidad. Al igual que la exactitud, sirve para hacerse una idea general de cómo de bueno es un clasificador. Puede no ser la mejor métrica para la detección de intrusiones, pues asume que precisión y sensibilidad tienen igual importancia. En nuestro caso esto no es así y por ello nos fijaremos primeramente en cada una por separado, dando especial importancia a la precisión. Consta de un parámetro beta para regular la importancia de cada una, aunque en este caso lo dejaremos en 1, de modo que ambas tengan la misma. De esa forma la Fscore se calcularía como:

$$Fscore = 2(P * S)/(P + S) \tag{6.4}$$

Se deseará por tanto un algoritmo de clasificación que maximice los valores de exactitud, precisión y sensibilidad. La importancia de cada una de estas métricas dependerá del tipo de sistema que se esté considerando. En general en un IDS se priorizará maximizar la precisión, debido a que una elevada tasa de falsos positivos puede implicar un exceso de trabajo por parte de los técnicos en seguridad a la hora de comprobar las alarmas. Es por ello que la prioridad relativa de una u otra métrica dependerá también de cómo de crítica es la seguridad del sistema. En nuestro caso, y siendo el objetivo principal la integración de la detección de anomalías con las técnicas basadas en firmas ya utilizadas, al objetivo principal será maximizar la precisión, de forma que cuando los técnicos en seguridad reciban una alerta generada por la IA, sepan que muy probablemente se trate de una alerta real y la examinen a conciencia.

También debería ser posible, para un mismo algoritmo, modificar su sensibilidad para configurar cuánta desviación respecto al comportamiento normal de la red debe presentar un paquete para generar una alarma. Incluso sería posible crear diferentes tipos de alarma en función del grado de desviación. Por ejemplo, podrían generarse alarmas medias, altas y críticas, con diferentes tiempos de revisión por parte de los técnicos.

Para la clasificación en múltiples clases (3 o más), que intentaremos en este estudio, estas métricas se calcularán para cada una de las clases. Solo en el caso de la exactitud trataremos de aproximar su valor para el clasificador en su conjunto. Puesto que la proporción de tráfico de red generado por los diferentes tipos de ataque es muy diferente, utilizaremos la **exactitud balanceada**, que otorga un peso diferente a los ejemplos de cada una de las clases en función de la presencia de ese tipo de ejemplos en el dataset. De esta forma las clases que aparezcan en menor proporción tendrán mayor peso, para que el echo de que el IDS sea capaz de clasificarlas o no afecte consecuentemente al valor de la métrica. No nos interesa un IDS que sea capaz de clasificar a la perfección los ataques de tipo Probe y DoS, que generan una cantidad de tráfico mucho mayor, si no es capaz de clasificar también de tipo R2L y U2R.

Por otro lado, a la hora de evaluar un algoritmo para su aplicación en un IDS, también es necesario analizar su **rendimiento computacional** a la hora de hacer un modelo de una red y de clasificar los eventos que se dan en ella. Las principales métricas utilizadas para determinarlo son el tiempo que tarda el algoritmo en crear un modelo del sistema (**Training Time**, TRT) y, sobre todo, en clasificar los paquetes una vez creado el modelo (**Testing Time**, TST) [1]. Aunque un algoritmo fuese capaz de clasificar a la perfección el tráfico de una red, un elevado tiempo de procesamiento podría hacerlo inútil para su aplicación en un IDS.

6.3. Datasets de tráfico de red. Selección de un dataset

Existen principalmente dos formas de representar la información sobre el tráfico de una red de ordenadores: el formato basado en paquetes y el formato basado en flujo [14].

- La **captura de paquetes** se hace generalmente copiando el tráfico de red que pasa por una determinada interfaz y almacenándolo en formato pcap. Este formato contiene toda la información sobre tanto las cabeceras como la carga de los paquetes en los diferentes protocolos del modelo OSI. Generalmente es suficiente con almacenar segmentos de la capa de transporte (TCP o UDP), que encapsulan los datos de las capas inferiores (IP, ICMP, etc.).

- Los **datos basados en flujos** contienen información de más alto nivel, inlcuyendo generalmente metadatos de las diferentes conexiones de red. Los datos basados en flujo suelen juntar información sobre los paquetes que comparten una serie de características durante un determinado espacio de tiempo, sin incluir su carga. En redes IP, la forma por defecto de agrupar los paquetes como pertenecientes a un mismo flujo es cuando comparten estas cinco propiedades: IP de origen, puerto de origen, IP de destino, puerto de destino y protocolo de la capa de transporte [14]. Los flujos pueden agruparse en un formato unidireccional o bidireccional. El un formato unidireccional se representan como un flujo diferente los paquetes enviados del host A al host B de los enviados desde B hacia A. En un formato bidireccional ambos flujos se agrupan juntos. Algunos formatos habituales basados en flujos son NetFlow, IPFX, sFlow y OpenFlow. Es posible convertir datos basados en paquetes en datos basados en flujos, pero no al revés, ya que este proceso conlleva una pérdida de información.

Existen otros formatos de datos que no corresponden a ninguna de estas dos categorías. El caso de mayor interés son los **datos basados en flujos enriquecidos** con datos relativos a los paquetes (tales como flags TCP) o con datos provenientes de logs de los hosts implicados (número de login fallidos, etc.). Un caso muy relevante es el del dataset KDD CUP 1999, uno de los más usados para IDS, que tiene este tipo de información. Según [11], es necesario añadir este tipo de información a la hora de recopilar datos basados en flujos para que puedan ser utilizados de forma eficiente para detectar y clasificar intrusiones. Además este tipo de datos ofrece una ventaja frente a los basados en paquetes, al ser más fáciles de obtener y más rápidos de procesar. También son más apropiados para detectar ataques en los que la frecuencia de envío de paquetes es menor, de acuerdo con los experimentos realizados para ese documento.

Por otro lado, los datasets pueden estar o no **etiquetados**. En el caso de los datasets para evaluación de sistemas de detección de intrusos, esto se traduce en si cada paquete o flujo de paquetes tiene asociado el tipo de tráfico al que corresponde: si es tráfico normal, si corresponde a un ataque, el tipo de ataque, etc. Solo un dataset que contenga estas etiquetas podrá ser utilizado en un proceso de aprendizaje supervisado.

Existen múltiples datasets con información de tráfico de red que pueden ser utilizados para la evaluación del rendimiento de un IDS. No todos están disponibles públicamente, para este estudio nos centramos en analizar algunos de los que sí lo están, y que han sido utilizados en experimentos similares [1] [14]. Las características de cada dataset (formato y etiquetado) determinarán la forma en que los algoritmos de machine learning los procesen para obtener información, y el rendimiento de los mismos a la hora de procesarlos. Estos fueron los datasets considerados:

- **KDD CUP 1999**: Basado en el dataset DARPA 1998/1999, creado por el MIT Lincoln Lab a partir de un entorno de red emulado, en el que se capturaron durante varias semanas tanto tráfico normal como ataques informáticos de distinto tipo (DoS, scanning, buffer overflow...). Ambos datasets han sido de los más utilizados en detección de intrusiones desde su publicación. De hecho el 79 % de los estudios revisados en [1] utilizan Datasets basados en DARPA. Tal y como se ha comentado anteriormente, KDD CUP 99 contiene datos basados en flujo enriquecidos con datos provenientes de logs de sistema y de los paquetes de red implicados en la comunicación. Contiene más de 20 tipos de ataques junto con tráfico normal y ya viene dividido en un subconjunto de entrenamiento para algoritmos de ML y otro de prueba. Se descarta por estar desactualizado, aunque poseen buenas características para el experimento. No obstante, el **Instituto Canadiense de Ciberseguridad (CIC)**, entidad que ofrece actualmente este dataset, posee disponibles otros más actualizados y de similares características.

- **NSL KDD**: Creado en 2009 como actualización del KDD CUP 99, solucionando algunos de los problemas de redundancia expuestos en [12], aunque sigue teniendo algunos de los problemas planteados por McHugh en [13]. A pesar de ello se sigue considerando un buen benchmark para evaluar diferentes métodos de detección de intrusiones y sigue siendo uno de los más utilizados en estudios recientes [1], por lo que ofrece la ventaja de que permite comparar los estudios realizados sobre él al estar su uso bastante estandarizado. Al igual que KDD CUP 99 tiene datos basados en flujos enriquecidos con información procedente de los hosts y los paquetes implicados

en la comunicación.

- **CIC-IDS2017**: Otro de los datasets ofrecidos por el CIC. Contiene información basada en flujos bidireccionales, con más de 80 características en total. Ha sido generado en una red emulada y viene dividido en diferentes ficheros correspondientes a los periodos de tiempo en los que se emularon los diferentes tipos de ataque. Contiene los tipos de ataque más recientes y comunes en redes reales en el momento de su publicación.
- **CSE-CIC-IDS2018**: Último de los datasets ofrecidos por el CIC en colaboración con el Communications Security Establishment (CSE). Ofrece tanto los datos en bruto (la captura del tráfico de red y los logs de los equipos) como los datos basados en flujo. Se centra en la división del dataset en perfiles que puedan combinarse en las proporciones deseadas para simular el comportamiento de una determinada red. Esto puede resultar bastante útil para que diferentes entidades experimenten con un dataset similar a su red, aunque en principio no ofrece ninguna ventaja para nuestro experimento.

De acuerdo con [11] es deseable utilizar datasets basados en flujo enriquecidos con otros datos. Por ello y por su carácter estándar hemos decidido utilizar el dataset **NSL KDD** para nuestro estudio. Además, se encuentra etiquetado, lo que es prácticamente un requisito para la detección de intrusiones.

6.4. El dataset NSL KDD. Estructura

A continuación expondremos brevemente la estructura y contenido del dataset, de acuerdo a la versión disponible en la página web del CIC y al análisis realizado en [20].

Como hemos dicho, el dataset contiene datos en formato basado en flujo, enriquecidos con datos de los paquetes y equipos implicados en la comunicación. El dataset tiene un total de 41 características además de la etiqueta y el nivel de dificultad. Estas características pueden dividirse en 4 grandes grupos. La tabla 6.1 muestra los atributos básicos de un flujo de paquetes. La tabla 6.2 muestra los atributos relativos al contenido de los paquetes que conforman el flujo, es decir, información relativa a la capa de aplicación. Por otro lado, las tablas 6.3 y 6.4 muestran información sobre otros flujos de paquetes de red que comparten alguna característica con el flujo al que corresponde la entrada. Esta información es de vital importancia para poder detectar ciertos tipos de ataque (escaneos de red, ataques DDoS...). En el caso de la tabla 6.3 se trata de información sobre otras conexiones temporalmente relacionadas con la dada, mientras que en la 6.4 están relacionadas por las direcciones IP o los números de puerto.

De esta forma pueden distinguirse características en formato de cadena de texto, otras en formato binario y otras en formato numérico. Más adelante será necesario tratar el dataset para tener un tipo de datos homogéneo con el que alimentar a los algoritmos de machine learning.

Finalmente, cada entrada del dataset incluye una etiqueta con la clasificación del flujo. Los flujos están clasificados como normales o como correspondientes a ataque informático. El dataset incluye flujos correspondientes a 40 tipos de ataques distintos, cada uno de los

Cuadro 6.1: Atributos básicos de flujo

Num.	Nombre	Descripción
1	Duration	Duración de la comunicación
2	Protocol_type	Protocolo de la capa de transporte utilizado en la comunicación (cadena de texto)
3	Service	Servicio de red utilizado por el cliente (cadena de texto)
4	Flag	Estado de la conexión (cadena de texto)
5	Src_bytes	Número de bytes enviados desde el origen al destino durante la conexión.
6	Dst_bytes	Número de bytes enviados desde el destino hacia el origen durante la conexión.
7	Land	Indica si las direcciones IP y números de puerto de origen y destino son iguales (1) o no (0)
8	Wrong_fragment	Número de fragmentos erróneos contabilizados en la conexión
9	Urgent	Número de paquetes con el bit de urgente activado contabilizados en la conexión

cuales puede englobarse en una de las categorías expuestas en la sección 2. Dicha correspondencia está recogida en la tabla 6.5. Por otro lado, se incluye una última característica que indica el grado de dificultad a la hora de clasificar cada entrada. Para este estudio obviaremos esta característica.

6.5. Tratamiento del dataset

6.5.1. División del dataset: Training Set y Test Set

Scikit-Learn proporciona herramientas para dividir automáticamente un dataset en dos subconjuntos aleatorios para aprendizaje y pruebas. Por defecto el Training Set comprenderá un 75 % del total, siendo el resto el 25 % restante el utilizado para evaluar la eficacia del clasificador. La tabla 6.6 muestra el número de instancias de cada clase tanto en Training Set como en Test Set utilizando dicha configuración.

6.5.2. Selección de características

No todas las características que ofrecen los datasets ofrecen la misma información a la hora de clasificar de los datos. De hecho, utilizar todas las características de los datos puede repercutir negativamente tanto en el rendimiento computacional como en la eficacia de los algoritmos de clasificación. Antes de la fase de entrenamiento de un dataset es habitual preprocesarlo para eliminar características redundantes o no relevantes.

Cuadro 6.2: Atributos relacionados con el contenido de los paquetes

Num.	Nombre	Descripción
10	Hot	Número de indicadores de peligrosidad detectados en el contenido de los paquetes, tales como: 'entering a system'
11	Num_failed_logins	Número de intentos de login fallidos
12	Logged_in	Indica se ha habido un login exitoso (1) o no (0)
13	Num_compromised	Número de posibles indicadores de compromiso detectados en el contenido de los paquetes
14	Root_shell	Indica si se ha obtenido una consola de administrador (1) o no (0)
15	Su_attempted	Indica se se ha intentado ejecutar el comando 'su root' durante la conexión (1) o no (0)
16	Num_root	Número de accesos y operaciones realizadas como root durante la conexión
17	Num_file_creations	Número de operaciones de creación de ficheros realizadas durante la conexión
18	Num_shells	Número de interpretes de comados utilizados durante la conexión
19	Num_access_files	Número de operaciones de acceso a ficheros de control
20	Num_outbound_cmds	Número de conexiones hacia el exterior realizadas en una sesión FTP
21	Is_hot_login	Indica si el login corresponde a un de los considerados potencialmente peligrosos (como los usuarios root o admin) (1) o si no (0)
22	Is_guest_login	Indica si el login es como invitado (1) o no (0)

Cuadro 6.3: Atributos sobre el tráfico temporalmente relacionado con el flujo

Num.	Nombre	Descripción
23	Count	Número de conexiones hacia el mismo host de destino que la actual en los últimos 2 segundos
24	Srv_count	Número de conexiones hacia el mismo servicio (puerto de destino) que la actual en los últimos 2 segundos
25	Serror_rate	Porcentaje de conexiones con los flag (4) s0, s1, s2 o s3 activados de entre las contabilizadas en count (23)
26	Srv_serror_rate	Porcentaje de conexiones con los flag (4) s0, s1, s2 o s3 activados de entre las contabilizadas en srv_count (24)
27	Rerror_rate	Porcentaje de conexiones con el flag (4) REJ activado de entre las contabilizadas en count (23)
28	Srv_rerror_rate	Porcentaje de conexiones con el flag (4) REJ activado de entre las contabilizadas en count (23) srv_count (24)
29	Same_srv_rate	Porcentaje de conexiones contra el mismo servicio (puerto de destino) que la actual, de entre las contabilizadas en count (23)
30	Diff_srv_rate	Porcentaje de conexiones contra diferentes servicios (puerto de destino) que la actual, de entre las contabilizadas en count (23)
31	Srv_diff_host_rate	Porcentaje de conexiones contra diferente host de destino que la actual, de entre las contabilizadas en srv_count (24)

Cuadro 6.4: Atributos sobre el tráfico relacionado con el host

Num.	Nombre	Descripción
32	Dst_host_count	Número de conexiones con la misma dirección IP de destino que la actual
33	Dst_host_srv_count	Número de conexiones con el mismo número de puerto que la actual
34	Dst_host_same_srv_rate	Porcentaje de conexiones hacia el mismo puerto de destino, de entre las contabilizadas en dst_host_count (32)
35	Dst_host_diff_srv_rate	Porcentaje de conexiones hacia diferentes puertos de destino, de entre las contabilizadas en dst_host_count (32)
36	Dst_host_same_src_port_rate	Porcentaje de conexiones desde el mismo puerto de origen, de entre las contabilizadas en dst_host_srv_count (33)
37	Dst_host_srv_diff_host_rate	Porcentaje de conexiones hacia diferentes host de destino, de entre las contabilizadas en dst_host_srv_count (33)
38	Dst_host_serror_rate	Procentaje de conexiones con los flag (4) s0, s1, s2 o s3 activados, de entre las contabilizadas en dst_host_count (32)
39	Dst_host_srv_serror_rate	Porcentaje de conexiones con los flag (4) s0, s1, s2 o s3 activados, de entre las contabilizadas en dst_host_srv_count (33)
40	Dst_host_rerror_rate	Porcentaje de conexiones con el flag (4) REJ activado, de entre las contabilizadas en dst_host_count (32)
41	Dst_host_srv_rerror_rate	Porcentaje de conexiones con el flag (4) REJ activado, de entre las contabilizadas en dst_host_srv_count (33)

Cuadro 6.5: Correspondencia entre los ataques recogidos en NSL KDD y su categoría

Categoría	**Ataques**	**Número**
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint	6
R2L	Guess Password, Ftp write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy, Xlock, Xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named	16
U2R	Buffer overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps	7
DoS	Back, Land, Neptune, Pod, Smurf, Teardrop, Apache2, Udpstorm, Processtable, Worm, Mailbomb	11

Cuadro 6.6: Número de instancias en el dataset NSL KDD dividido

Clase	**Instancias en Training Set**	**Instancias en test set**
Normal	57774	19280
Probe	10528	3549
R2L	2892	988
U2R	90	29
DoS	40103	13284

La selección de características no siempre se realiza en el campo de la detección de intrusiones [2], ya que muchos datasets ofrecen sus datos en formatos que muestran solo las características consideradas relevantes a la hora de detectar ataques. No obstante, en este documento consideraremos algunos métodos para eliminar características irrelevantes y aplicaremos alguno de ellos si resulta pertinente. Se ha observado que, en nuestro caso, la selección de características mejora notablemente el tiempo de entrenamiento (TRT) de algunos clasificadores. Además, si es la adecuada, también mejora la eficacia en algunos casos. Por ello, se pretende realizar también un estudio de cómo afectan el número de características y el método para seleccionarlas a la eficacia de los clasificadores.

Se aplicarán dos métodos automáticos de selección de características que permitan procesar el dataset etiquetado y determinar qué características del mismo son menos relevantes para clasificar la naturaleza de los eventos de la red. El primer método elegido es **Recursive Feature Elimination (RFE)**. El algoritmo realiza varias iteraciones en las que calcula la relevancia las características del dataset. Comienza haciéndolo con todas y elimina la menos relevante en cada iteración. Tiene dos parámetros importantes: el clasificación utilizado para evaluar la relevancia de las características (estimador) y el número de características deseado. En nuestro caso el método de clasificación será Decision Trees, al ser uno de los clasificadores más rápidos y al estar muchos de los métodos compuestos que vamos a utilizar basados en él (lo veremos más adelante). Para determinar el número de características óptimo una opción sería utilizar una variación de RFE que permita hacerlo automáticamente. RFECV es una versión de RFE que realiza validación cruzada utilizando los datos del dataset para determinar el número óptimo de características, de forma que no es necesario indicarlo de antemano. No obstante, queremos evaluar el rendimiento de los clasificadores para cada característica y método de selección, por lo que esto no nos sirve, utilizaremos RFE para cada uno de los casos. El otro método de selección de características es el método estadístico **Análisis de Componentes Principales (PCA)**. La idea de este método es convertir el conjunto inicial de características, probablemente correlacionadas, en un conjunto menor de características sin correlación lineal que conserven la mayor cantidad de información posible respecto al conjunto original. Tiene la ventaja de que no es necesario seleccionar un clasificador como estimador de la relevancia de cada característica, por lo que no se introducen sesgos al aplicarlo. Tiene la desventaja de que no tiene en cuenta el problema de clasificación que se está abordando a la hora de seleccionar las características.

Se aplicará tanto RFE como PCA al dataset para seleccionar 5, 10, 15, 20 y 25 características. Se evaluará la eficacia media de los clasificadores en todos los casos respecto a una misma métrica, de forma que se determine el número de características óptimo y el mejor método para seleccionarlas. La métrica respecto a la que los compararemos será la exactitud balanceada media de todos los clasificadores. Una vez seleccionado un número de características y un método de selección, procederemos a evaluar la eficacia de los clasificadores en ese caso, teniendo ya en cuenta todas las métricas expuestas anteriormente.

6.5.3. Preprocesamiento

Los algoritmos de clasificación implementados en ScikitLearn necesitan procesar datos numéricos. Por tanto, se sustituirán todas las características del dataset que se encuentren en formato string (incluyendo la etiqueta) por un número entero único para cada cadena.

Se guardarán todas las características en formato float de 32 bits. Además, se sustituirán todos los NaN que se puedan encontrar por números.

Por otro lado, algunos de los clasificadores necesitan que los datos con los que trabajan hayan sido escalados o normalizados, por lo que se tratará a los datos de la forma pertinente en cada caso.

Además, se agruparán las etiquetas correspondientes a todos los grupos para sustituir las correspondientes a ataques concretos por la familia a la que pertenecen, de acuerdo a la información recogida en la tabla 6.5.

6.6. Algoritmos de ML a estudiar

Para este estudio solo se considerarán aquellos algoritmos que han obtenido mejores resultados en los estudios realizados hasta el momento, para lo cual nos fijaremos en la información recopilada en [2], [11] y, sobre todo, en [1].

Clasificadores simples

Estos algoritmos son los seleccionados:

- **Decision Trees (DT)** [9]. Se trata de una familia de algoritmos que clasifican los datos realizando una serie de decisiones en función de ellos. Dicha secuencia de decisiones es representada mediante una estructura de árbol. La clasificación de un dato se realiza desde el nodo raíz hasta un nodo hoja adecuado, donde cada nodo hoja corresponde a una categoría [7]. El algoritmo implementado por Scikit-Learn es **CART (Classification and Regression Trees)**. Se trata de una versión modificada de los algoritmos ID3 y C4.5. Este algoritmo crea un árbol de decisiones binarias durante la fase de entrenamiento supervisado, que se interpretan como reglas de tipo if-then. En cada nodo se toma una decisión en función del valor de un determinado atributo. Las decisiones y su orden se determinan en función de la relevancia de la misma para clasificar el dato durante la fase de entrenamiento.

- **Support Vector Machines (SVM)**. Se trata de uno de los enfoques más utilizados en cuanto a aprendizaje automático supervisado [8]. Además, es uno de los que han ofrecido mejores resultados en cuanto a su eficacia a la hora de clasificar intrusiones. Está diseñado para la clasificación binaria y permite la clasificación de datos con un número de características muy elevado. Se basa en el cálculo de un hiperplano. Definimos **hiperplano** como un subespacio con una dimensión menos que el espacio de los datos de entrada: para un plano sera una recta que lo divide en dos, para una recta sería un punto, etc. El proceso de aprendizaje (supervisado) mediante SVM consiste en el cálculo de un hiperplano que separe todos los datos de la muestra de forma que se intente que todos los datos se encuentren en el lado correcto del hiperplano (clasificados correctamente) y lo más lejos posible de él [16]. Para realizar este cálculo de una forma eficiente se utilizan **funciones kernel**, que permiten obtener una medida del grado de similitud de dos puntos en un espacio de mayor dimensionalidad al de los datos dados, sin tener que calcular las coordenadas de los puntos en dicho espacio. En la clasificación mediante SVM pueden usarse diferentes funciones kernel (lineal, gaussiano, polinómico...) con diferentes parámetros.

La idoneidad de una u otra dependerá en gran medida de los datos de entrada. En cuanto a los demás parámetros del clasificador, resulta importante el **parámetro de penalización C** o coste. C determina el grado de importancia de los puntos mal clasificados. A mayor C implica que se da más importancia a clasificar correctamente todos los puntos del Training Set que a dejar espacio en los modelos de cada clase para la clasificación de futuros datos. La elección de estos parámetros la haremos de forma automática con herramientas de validación cruzada, fijándonos también en los parámetros utilizados en otros estudios como [18] y [19]. Probamos una serie de parámetros considerados en estos estudios utilizando la herramienta de validación cruzada RandomizedSearchCV, de forma que los considerados óptimos fueron los siguientes: kernel gaussiano, C=5000 y gamma = 0.1. Más adelante se verá la selección de características en más detalle.

- **Artificial Neural Networks (ANN)**. Se trata de una familia de algoritmos de machine learning que procesan la información mediante unidades que imitan el comportamiento de las neuronas del cerebro humano. Cada una de estas unidades (llamadas **neuronas**) toma un conjunto de datos como entrada y produce un único dato de salida, que podrá ser utilizado como entrada por otras unidades de procesamiento. Vamos a centrarnos en la arquitectura más utilizada: las redes neuronales feedforward donde la red de neuronas puede representarse como un grafo dirigido sin ciclos, de forma que cada nodo es una neurona y cada enlace representa que la salida de una neurona es la entrada de otra. Se considera por simplificar que la red está formada por capas, de forma que las neuronas de la capa i reciben como entrada la salida de las de la capa i-1 (excepto en la capa 0) y pasan su salida a la capa i+1 (excepto en la última capa). La primera capa tiene n+1 neuronas, siendo n la dimensionalidad de los datos de entrada. La salida de cada neurona de la primera capa tras alimentarla con el vector $\mathbf{x}$ es cada uno de los atributos del vector, excepto en el vector n+1 que siempre es 1. Las neuronas de las capas intermedias o capas ocultas reciben como entrada la suma ponderada de las salidas de las neuronas de la capa anterior que conectan con ellas, y calculan la salida aplicando una función (**función de activación**) a ese valor. Se denomina **arquitectura de una red neuronal** al conjunto de sus nodos, sus enlaces y la función de activación aplicada en sus neuronas. El proceso de aprendizaje se realiza aplicando diferentes pesos a los enlaces de una arquitectura fija. Cada combinación de posibles pesos en los enlaces es una hipótesis. Durante el proceso de aprendizaje supervisado se irán ajustando para que la capa de salida, de un solo nodo, produzca la salida deseada [16]. Una de las arquitecturas más utilizadas es el **Multilayer Perceptron (MLP)** [2], que será la utilizada para este estudio.

- **Naive Bayes Classifier** (NB). Se trata de un modelo generativo. Esto quiere decir que, al contrario que en los casos anteriores, se asume que los datos de entrada pueden ser representados por un modelo y unos parámetros, de modo que la fase de aprendizaje consiste en intentar encontrar dichos parámetros [16]. El problema consiste en que suele ser más complicado determinar la distribución que siguen los datos de entrada que crear un clasificador adecuado sin hacerlo, como ocurría en los algoritmos explicados anteriormente. Naive Bayes trata de una simplificación del **Optimal Bayes Calssifier** (basado en el **Teorema de Bayes**) en el que se reduce significativamente el número de parámetros que hay que estimar mediante una suposición. En el caso de Naive Bayes asumimos que, dada una etiqueta, las

características del vector de entrada son independientes entre sí. Hacemos esta suposición sabiendo que en la mayoría de los casos es incorrecta (en la detección de intrusiones desde luego lo es), pero que simplifica enormemente la generación de un modelo probabilístico. Es por ello que el clasificador se llama así (*naive* en inglés significa ingenuo). Los parámetros se estiman mediante el **principio de máxima verosimilitud** (maximun likelihood principle).

Será necesario determinar los parámetros óptimos para estos algoritmos.

Clasificadores compuestos

Los clasificadores compuestos son aquellos en los que se utilizan los resultados de varios clasificadores simples, ya sean del mismo o de diferente tipo, para realizar la clasificación final en función de los resultados de éstos.

Existen diferentes métodos para de poner en común los resultados de los clasificadores firmes para tomar una decisión final.

- **Bagging**. Se construyen diferentes modelos sobre diferentes porciones del dataset original. Generalmente se hace con clasificadores simples del mismo tipo.
- **Boosting**. Se construyen diferentes modelos. El primero aprende a clasificar los datos, mientras que los siguientes tratan de aprender a corregir los errores de predicción del anterior.
- **Voting**. Se construyen diferentes modelos, generalmente con clasificadores simples de distinto tipo, y se combinan los resultados de los mismo mediante un proceso de votación o algún cálculo estadístico sencillo.

Para este estudio consideraremos algún ejemplo de cada uno de estos métodos, de los implementados por Scikit Learn. Estos han sido los que hemos seleccionado.

- **Random Forest** (RF). Se trata de clasificador que consiste en una colección de árboles de decisión (DT), en la que cada árbol es construido aplicando un algoritmo a un subconjunto del Training Set a partir de un vector aleatorio, de forma que este vector determina la forma en que se genera el árbol. Se utiliza por tanto una técnica de bagging. Los resultados finales se obtienen mediante una votación sobre las predicciones de cada uno de los árboles.
- **Adaptive Boosting** o **AdaBoost**. El clasificador AdaBoost se basa en la creación de un conjunto de clasificadores débiles, típicamente árboles de decisión (DT). En primer lugar se construye el primer árbol de acuerdo con un algoritmo de clasificación (por ejemplo ID3 o CART) a partir de los datos del Training Set. Después se evalúa la eficacia de dicho modelo y se crea un segundo árbol dando mayor peso a aquellos elementos del Training Set que no han sido clasificados correctamente por el primero. Este proceso se va repitiendo sucesivamente.
- **Gradient Boosting Machine** (GBM). Este clasificador se diferencia de AdaBoost en la forma en la que trata de corregir el error de los clasificadores simples. Mientras AdaBoost lo hace ajustando los pesos de los elementos del Training Set. GBM lo hace ajustando los parámetros de la **función de pérdida**. En clasificación, una

función de pérdida es aquella que calcula el valor de la penalización debida a la clasificación errónea de un dato. De esta forma se trata el problema de clasificación como un problema de optimización en el que se pretende minimizar la pérdida del modelo añadiendo clasificadores simples al conjunto. Pueden utilizarse diferentes funciones de pérdida estándar para esto, o puede definirse una específica.

- **Voting Classifier** (VC). Se trata de una de las formas más simples de combinar varios clasificadores. Simplemente se determina la clasificación final de cada ejemplo mediante el voto por mayoría de todos los clasificadores. También puede predecirse la clase calculando la media de las probabilidades otorgadas por cada clasificador simple a cada clase, y eligiendo aquella con una media superior. Esto será lo que haremos en este estudio. Pueden usarse clasificadores de distinto tipo.

6.7. Implementación

La implementación constará de un **script principal** en Python3 basado en las librerías de Scikit Learn y en una estructura de directorios. Además, hará uso de **un script auxiliar por cada clasificador** evaluado y de otro **script auxiliar para la extracción de gráficas**. Todos los ficheros necesarios se encuentran disponibles en GitLab [1].

El script principal, **Test.py**, leerá los datos y ejecutará otros scripts auxiliares contenidos en estos directorios, el diagrama de actividad del script se encuentra representado en la figura 6.2. El propio script creará, si no existen, los directorios de salida donde guardará los resultados de la evaluación de los clasificadores. Cada ejecución del script sobrescribirá estos datos. Para replicar el experimento realizado tan solo es necesario clonar el repositorio de Gitlab y ejecutar el script Test.py.

La **estructura de directorios** es la siguiente, tal y como se encuentra disponible en GitLab y representada en la figura 6.1. Un directorio base ./ con el script principal, que realiza todo el estudio en cuanto a los clasificadores. Dos subdirectorios necesarios: el subdirectorio ./datos deberá contener el dataset KDD NSL en formato CSV y el subdirectorio ./clasificadores, con un script para cada uno de los clasificadores que se pretenda evaluar, siguiendo en todos casos una misma estructura. Además, el propio script principal creará, si no existen, 3 subdirectorios de salida donde escribirá datos. En ./modelos se guardan los modelos de Scikit Learn que deberán utilizarse en el futuro. En ./resultados se guardarán los ficheros con las métricas de rendimiento resultantes del estudio. En la raíz habrá un fichero que permita comparar la eficacia de los clasificadores según la selección de características, y una serie de subdirectorios con las métricas correspondientes al estudio con cada subconjunto de características. Finalmente, en ./gráficos se replicará la misma estructura para almacenar las gráficas y tablas que permitan visualizar estos datos.

6.7.1. Selección de parámetros

Para poder implementar el script primeramente debemos **seleccionar los parámetros** adecuados para todos los clasificadores. En nuestro caso, los clasificadores en los que era necesario considerar algún parámetro a priori son ANN y SVM.

[1] En el repositorio público `https://gitlab.inf.uva.es/gonvald/tfg-gva-mlids2-doc`, en el subdirectorio TestTFG.

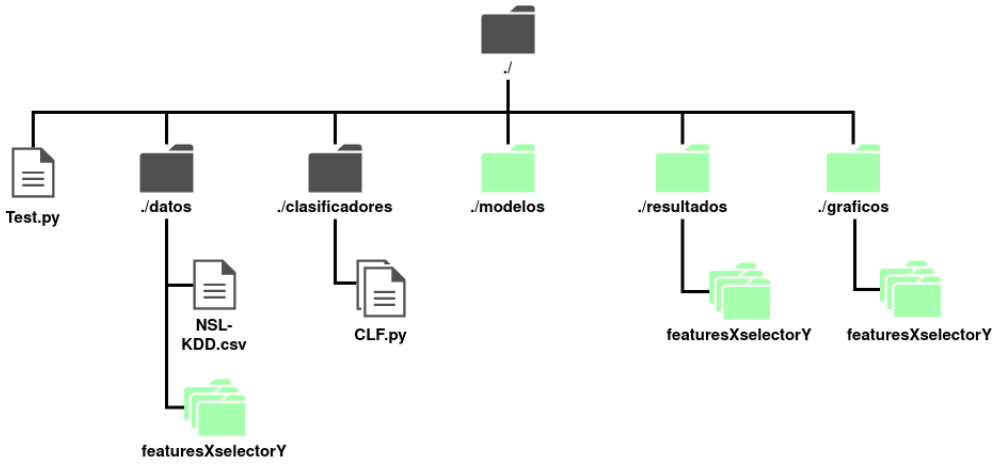

Figura 6.1: Árbol de directorios utilizado por el script. En un color más claro, los directorios creados en tiempo de ejecución.

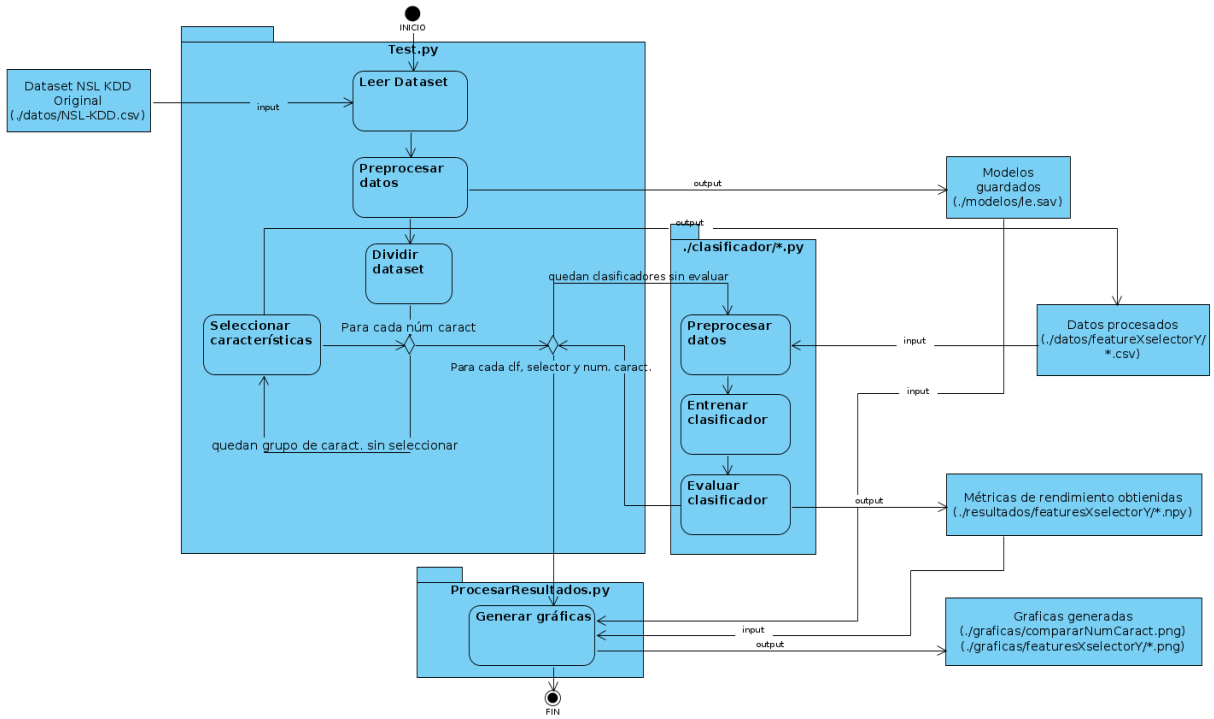

Figura 6.2: Diagrama de actividad del script principal que realiza el estudio práctico.

En el caso de **ANN** el parámetro a determinar es la estructura de la red de neuronas. Se optó por una estructura clásica de tres capas, seleccionando el **número de neuronas de la capa oculta** de acuerdo a lo expuesto en [19]. De esta forma y, como se va a evaluar la eficacia de los clasificadores con diferentes números de características en la entrada, el número de nodos de la capa oculta será igual a éste.

En el caso de **SVM** utilizaremos una las herramientas que nos proporciona Scikit Learn para realizar validación cruzada, **RandomizedSearchCV**, mediante la cual probamos diferentes tipos de **kernel** (lineal, polinómico, gaussiano, sigmoide), diferentes valores para el **parámetro de penalización C** (10, 100, 1000, 5000, 10000) y diferentes valores para el **coeficiente de los kernels** RBF, polinómico y sigmoide, gamma (0.00001, 0.001, 0.1). Estos valores de prueba se obtuvieron revisando la bibliografía y los valores típicamente

utilizados en detección de intrusiones. Mediante esta herramienta de búsqueda aleatoria se optimizaron estos parámetros en cuanto a la exactitud del clasificador, no en cuanto al tiempo empleado, quedando como resultado la selección del kernel RBF (Radial Basis Function kernel), C=5000 y gamma = 0.1. El kenrel RBF parecía ajustarse más a la distribución de los datos. Además, permite su tratamiento en un tiempo de entrenamiento mucho menor que los otros dos kernels que ofrecían mejores resultados: el polinómico y el lineal. Al valor del parámetro de penalización ha sido seleccionado automáticamente, dando un valor bastante más alto que el utilizado en la bibliografía consultada [19].

6.7.2. Pasos realizados por el script

Una vez se ha creado el script asociado a cada uno de los clasificadores, puede ejecutarse el script principal. Los pasos seguidos por el programa son los siguientes.

- **Carga del dataset y procesamiento del dataset**. Se abre el fichero CSV con el dataset KDD NSL situado en ./datos, bajo el nombre 'NSL-KDD.csv'. Se codifican las etiquetas y las características que están en formato string para que pueda aplicarse el dataset a los algoritmos de machine learning, que requieren datos numéricos. Se agrupan las etiquetas del dataset. Se guarda el modelo utilizado para codificarlos para poder recuperar más tarde los valores originales.

- **Selección de características**. Se seleccionan las características con mayor información de forma que queden 5, 10, 15, 20 y 25 características; utilizando dos algoritmos de selección de características, RFE y PCA, de acuerdo a lo expuesto en la sección 6.5.2. Se guardan versiones del dataset con las características seleccionadas en subdirectorios de la carpeta ./datos. Un subdirectorio por número de características evaluado y por algoritmo de selección.

- **Evaluación de clasificadores**. Se ejecutan los scripts auxiliares del directorio ./clasificadores, y se evalúa cada clasificador para cada uno de los subconjuntos de características seleccionados, guardando las métricas extraídas en disco, en el subdirectorio ./resultados. La evaluación de cada clasificador consta de tres pasos. En primer lugar, se preprocesa el dataset seleccionado, escalando o normalizando los datos según sea necesario para cada clasificador. En segundo lugar, se entrena el clasificador (fase de entrenamiento) con el Training Set. Seguidamente, se trata de predecir las clases del Test Set (fase de clasificación) y se extraen las métricas de rendimiento, guardándolas en disco en el subdirectorio correspondiente (de nuevo, uno para cada selector y número de características).

- **Generación de Gráficos**. Se llama al script auxiliar de extracción de gráficas, del que hablaremos a continuación.

6.7.3. Extracción de resultados

Se ha elaborado un script adicional que permite extraer gráficas y tablas a partir de los datos generados por el test. Se encuentra en la carpeta raíz bajo el nombre **ProcesarResultados.py**.

Este script leerá los ficheros creados por cada clasificador en la carpeta ./resultados y sus subdirectorios; y elaborará tablas y gráficas a partir de ellos, lo que nos facilitará la comparación de los diferentes clasificadores. Para la creación de las diferentes gráficas

usaremos las librerías de Python de Plotly. El formato de las tablas y gráficas será el ofrecido por éstas.

Capítulo 7

Análisis de los resultados

En primer lugar vamos a **analizar la influencia del número de características y su método de selección** en la eficacia de los clasificadores. La figura 7.1 muestra la exactitud balanceada media de los algoritmos de clasificación para cada número, de acuerdo a lo definido en 6.2; siendo los métodos de selección RFE y PCA. Es decir, se ha obtenido la exactitud balanceada de todos los clasificadores para cada número de características y método de selección y se ha obtenido la media de todas ellas, para tener una única métrica con la que comparar todos los casos.

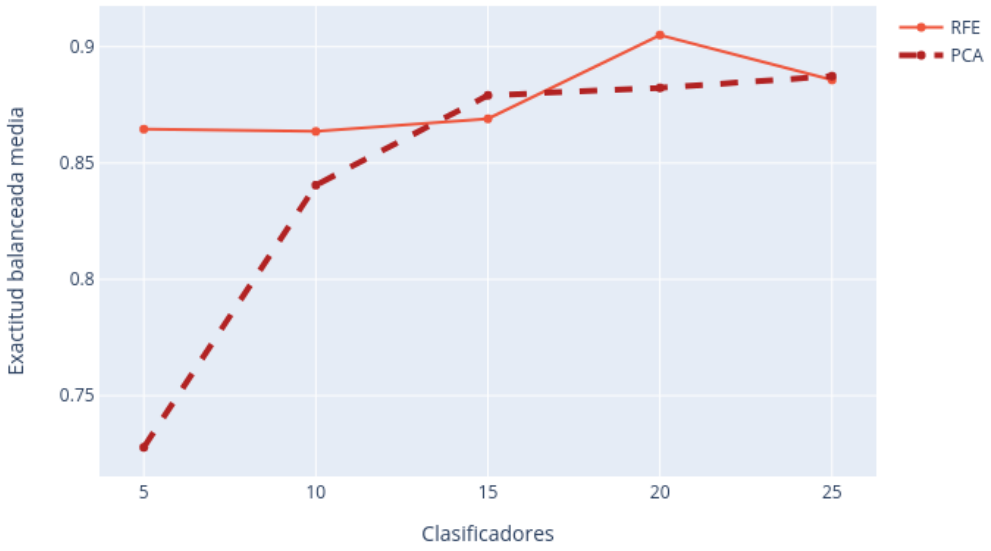

Figura 7.1: Exactitud balanceada media de los clasificadores en función del número de características, seleccionándolas mediante RFE

Observando las gráficas vemos que en general la exactitud aumenta conforme aumenta el número de características, especialmente usando PCA. Revisando los datos vemos que los valores de exactitud suelen ser mayores para RFE en la mayoría de los casos, especialmente para números de características pequeños. En los casos en los que es mejor para PCA apenas hay diferencia respecto a RFE. Esto también ocurre para aquellos al-

goritmos no basados en Decision Trees, que es el estimador utilizado para seleccionar las características. También hay diferencias respecto a los tiempos de entrenamiento. Para un mismo número de características suelen ser mayores en el caso de PCA. Esto puede deberse a que los clasificadores tienen más dificultades para crear un modelo utilizando características menos relevantes para el problema. De esta forma, **consideramos que el mejor método de selección es RFE, y un buen número de características 20**, para el que en este caso se obtuvo la mayor exactitud. La tabla 7.1 muestra los atributos seleccionados de entre los expuestos anteriormente.

Una vez fijados estos parámetros, podemos **analizar la eficacia de todos los clasificadores** respecto a las métricas descritas en la sección 6.2. Nos centraremos principalmente en cuatro métricas: la **exactitud** (balanceada, se utiliza como indicador de la eficacia general de un clasificador), la **precisión** (para tratar de minimizar el número de falsos positivos por parte de un posible IDS basado en ML) y los **tiempos de entrenamiento y clasificación** (para que sea posible aplicar el clasificador en un IDS en tiempo real).

Figura 7.2: Exactitud balanceada de cada clasificador, para 20 características seleccionadas mediante RFE

Comenzamos con la **exactitud**, que es la que nos da una idea general de cómo de bien ha clasificado cada algoritmo las muestras en las 5 categorías. La figura 7.2 nos da una idea su valor para cada uno de los clasificadores. De esta forma vemos que los clasificadores que obtuvieron una mayor exactitud fueron ANN, RF, y GBM. No obstante revisando los datos vemos que ANN no logró clasificar ninguna instancia de los ataque U2R, que aparecen en menor proporción, lo que lo descarta como un buen clasificador. Más adelante lo veremos más en detalle.

Nos fijamos ahora en la **precisión**, la fracción de positivos que realmente lo eran. Una precisión baja indicaría que gran cantidad de flujos de red han sido clasificados

Cuadro 7.1: Atributos seleccionados por RFE

Num.	Nombre	Descripción
1	Duration	Duración de la comunicación
2	Protocol_type	Protocolo de la capa de transporte utilizado en la comunicación
3	Service	Servicio de red utilizado por el cliente
4	Flag	Estado de la conexión
5	Src_bytes	Número de bytes enviados desde el origen al destino durante la conexión.
6	Dst_bytes	Número de bytes enviados desde el destino hacia el origen durante la conexión.
11	Num_failed_logins	Número de intentos de login fallidos
22	Is_guest_login	Indica si el login es como invitado (1) o no (0)
23	Count	Número de conexiones hacia el mismo host de destino que la actual en los últimos 2 segundos
30	Diff_srv_rate	Porcentaje de conexiones de (23) contra diferentes servicios (puerto de destino) que la actual.
32	Dst_host_count	Número de conexiones con la misma dirección IP de destino que la actual
33	Dst_host_srv_count	Número de conexiones con el mismo número de puerto que la actual
34	Dst_host_same_srv_rate	Porcentaje de conexiones de (32) hacia el mismo puerto de destino.
35	Dst_host_diff_srv_rate	Porcentaje de conexiones de (32) hacia diferentes puertos de destino.
36	Dst_host_same_src_port_rate	Porcentaje de conexiones de (33) desde el mismo puerto de origen.
37	Dst_host_srv_diff_host_rate	Porcentaje de conexiones de (33) hacia diferentes host de destino.
38	Dst_host_serror_rate	Porcentaje de conexiones de (32) con los flag (4) s0, s1, s2 o s3 activados.
39	Dst_host_srv_serror_rate	Porcentaje de conexiones de (33) con los flag (4) s0, s1, s2 o s3 activados.
40	Dst_host_rerror_rate	Porcentaje de conexiones de (32) con el flag (4) REJ activado.
41	Dst_host_srv_rerror_rate	Porcentaje de conexiones de (33) con el flag (4) REJ activado.

como ataques cuando no lo eran, o que han sido clasificados como el tipo incorrecto de ataque. No tiene sentido tener en cuenta la precisión global de cada clasificador, si no que nos debemos centrar en revisar la precisión de cada clasificador para cada clase. Más concretamente, a la hora de clasificar los 4 tipos de ataques, que es lo que generaría falsas alarmas y una sobrecarga de trabajo para los técnicos de seguridad. Por tanto nos fijaremos en las tablas 7.3 , 7.4 y 7.5; que nos muestran las métricas de rendimiento para cada clase de los clasificadores ANN, RF y GBM, respectivamente. Aquí vemos que ANN no ha podido clasificar ataques de tipo U2R en ningún caso, y que su sensibilidad y su precisión a la hora de clasificar los de tipo R2L también son significativamente más bajas.

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.987	0.996	0.979	13284.0
Normal	0.982	0.976	0.988	19280.0
Probe	0.971	0.964	0.978	3549.0
R2L	0.881	0.896	0.865	988.0
U2R	0.0	0.0	0.0	29.0

Figura 7.3: Métricas de rendimiento del clasificador **ANN**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.999	0.999	0.999	13284.0
Normal	0.996	0.995	0.997	19280.0
Probe	0.997	0.997	0.997	3549.0
R2L	0.95	0.965	0.936	988.0
U2R	0.524	0.846	0.379	29.0

Figura 7.4: Métricas de rendimiento del clasificador **RF**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.998	0.998	0.998	13284.0
Normal	0.994	0.993	0.995	19280.0
Probe	0.986	0.984	0.988	3549.0
R2L	0.942	0.971	0.914	988.0
U2R	0.731	0.826	0.655	29.0

Figura 7.5: Métricas de rendimiento del clasificador **GBM**, para 20 características seleccionadas mediante RFE

Finalmente, nos fijamos en los **tiempos de entrenamiento** y **clasificación**, mostrados en las figuras 7.6 y 7.7, respectivamente. Los algoritmos con mayor tiempo de entrenamiento han sido, con gran diferencia, ANN y VC; seguidos muy de lejos por GBM y SVM. Este tiempo de entrenamiento puede no condicionar demasiado la aplicación del algoritmo a un IDS, si no se pretende entrenar el algoritmo regularmente. En ese caso, solo afectaría al tiempo de instalación y puesta en marcha del IDS. En cuanto a los tiempos de clasificación, el único con un tiempo significativamente más alto es SVM, con 2.8 segundos para las los 37130 muestras del Test Set. Esto podría hacerle un mal candidato para su

aplicación en un IDS en tiempo real. De los clasificadores restantes, los compuestos son los que tienen tiempos de clasificación más altos, siendo de hasta 0.3 segundos en el caso de RF. No obstante, no se considera un tiempo que impida en ningún modo su aplicación en un IDS.

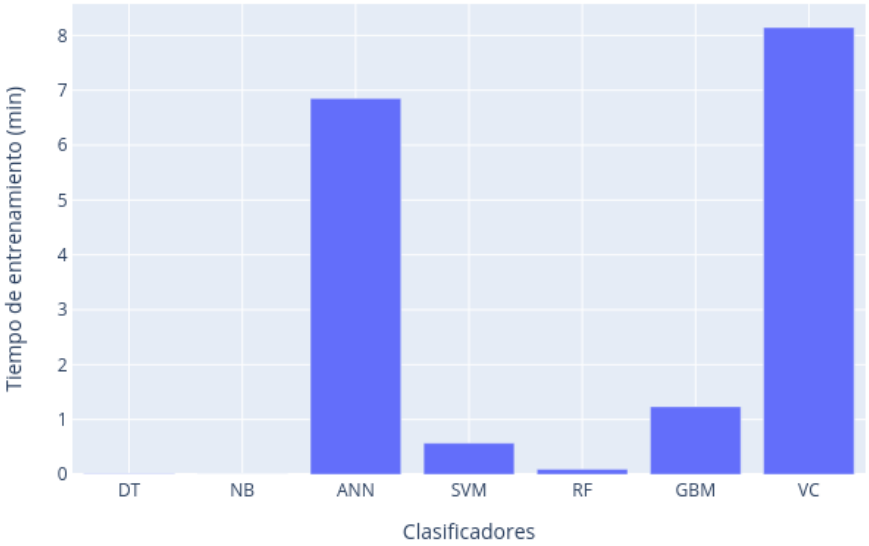

Figura 7.6: Tiempo de entrenamiento de cada clasificador, en minutos. Para 20 características seleccionadas mediante RFE

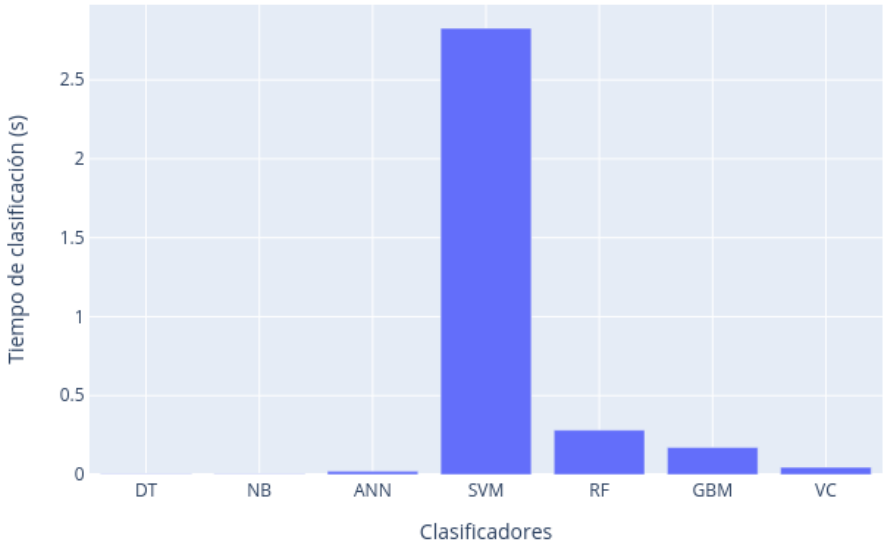

Figura 7.7: Tiempo de clasificación de cada clasificador, en segundos. Para 20 características seleccionadas mediante RFE

Capítulo 8

Conclusiones y trabajo futuro

De acuerdo con el análisis realizado, **Random Forest (RF)** y **Gradient Boosting Machine (GMB)** son los mejores algoritmos a la hora de detectar intrusiones. Siempre teniendo en cuenta que esto ocurre para el formato de datos del dataset NSL KDD y para las características seleccionadas por el algoritmo RFE, pero puede no ser aplicable en otras situaciones. Otros clasificadores, especialmente DT, también obtuvieron valores adecuados en cuanto a exactitud balanceada y precisión. Además, hay que recordar que clasificadores como SVM obtuvieron resultados muy prometedores en otros estudios sobre detección de intrusiones. Se considera que muchos de algoritmos deberán seguir teniéndose en cuenta en futuros estudios similares, e incluso podrán ofrecerse como opciones en futuras implementaciones de un IDS.

Como trabajo posterior, merecería la pena aplicar este mismo enfoque, con los mismos parámetros, a un dataset con otro formato de datos, especialmente si contiene otro tipo de información como las direcciones IP concretas, para ver en qué medida varía el rendimiento de los clasificadores y el número de características necesarias. También sería interesante aplicar en ambos casos un enfoque híbrido, en el que en un primer lugar se distingan los eventos legítimos de los ataques. Hacer esta primera distinción de forma aislada puede resultar conveniente ya que de esta forma un clasificador deberá modelar el tráfico normal de cada red sin tener que ser capaz de distinguir entre tipos de ataque. Así podemos controlar realmente la precisión y la sensibilidad del sistema de detección independientemente de si luego es capaz de clasificar o no los ataques. Un ejemplo en el que basarse podría ser el adoptado en Al-Yaseen et al. (2016) [10]. Dicho modelo, aunque no obtuvo valores de exactitud significativamente más elevados que otros clasificadores, sí que redujo notablemente la tasa de falsos positivos. Como hemos dicho, esto se considera de vital importancia a la hora de comenzar a implementar sistemas de detección de anomalías que resulten útiles en la práctica para los técnicos en seguridad.

Una vez realizado este estudio el siguiente paso podría ser tratar de utilizar la información obtenida para integrar un sistema de detección de anomalías con alguno de los IDS comerciales ya existentes. Por ejemplo, podría tratarse de crear un script con el que tratar los logs de Zeek y evaluar sus ventajas como complemento a otras formas de detección de intrusiones.

Apéndice A

Métricas de rendimiento obtenidas

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.998	0.998	0.999	13284.0
Normal	0.995	0.995	0.995	19280.0
Probe	0.994	0.994	0.995	3549.0
R2L	0.944	0.947	0.941	988.0
U2R	0.525	0.5	0.552	29.0

Figura A.1: Métricas de rendimiento del clasificador **DT**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.828	0.761	0.908	13284.0
Normal	0.872	0.87	0.873	19280.0
Probe	0.388	0.553	0.298	3549.0
R2L	0.045	1.0	0.023	988.0
U2R	0.0	0.0	0.0	29.0

Figura A.2: Métricas de rendimiento del clasificador **NB**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.987	0.996	0.979	13284.0
Normal	0.982	0.976	0.988	19280.0
Probe	0.971	0.964	0.978	3549.0
R2L	0.881	0.896	0.865	988.0
U2R	0.0	0.0	0.0	29.0

Figura A.3: Métricas de rendimiento del clasificador **ANN**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.988	0.997	0.979	13284.0
Normal	0.981	0.972	0.991	19280.0
Probe	0.964	0.987	0.943	3549.0
R2L	0.918	0.908	0.929	988.0
U2R	0.326	0.5	0.241	29.0

Figura A.4: Métricas de rendimiento del clasificador **SVM**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.999	0.999	0.999	13284.0
Normal	0.996	0.995	0.997	19280.0
Probe	0.997	0.997	0.997	3549.0
R2L	0.95	0.965	0.936	988.0
U2R	0.524	0.846	0.379	29.0

Figura A.5: Métricas de rendimiento del clasificador **RF**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.998	0.998	0.998	13284.0
Normal	0.994	0.993	0.995	19280.0
Probe	0.986	0.984	0.988	3549.0
R2L	0.942	0.971	0.914	988.0
U2R	0.731	0.826	0.655	29.0

Figura A.6: Métricas de rendimiento del clasificador **GBM**, para 20 características seleccionadas mediante RFE

Clase\Metrica	Fscore	Precision	Sensibilidad	Total
DoS	0.998	0.997	0.999	13284.0
Normal	0.993	0.992	0.995	19280.0
Probe	0.987	0.987	0.988	3549.0
R2L	0.935	0.959	0.911	988.0
U2R	0.176	0.6	0.103	29.0

Figura A.7: Métricas de rendimiento del clasificador **VC**, para 20 características seleccionadas mediante RFE

Bibliografía

[1] Fadi Salo, Mohammadnoor Injadat, Ali Bou Nassif, Abdallah Shami, Aleksander Essex. *Data Mining Techniques in IntrusionDetection Systems: A Systematic Literature Review*. 2018. IEEE Communications Surveys & Tutorials.

[2] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, Wei-Yang Lin (2009). *Intrusion detection by machine learning: A review*. Elsevier.

[3] ATT&CK Matrix for Enterprise. Retrieved from `https://attack.mitre.org`

[4] Stallings, W. (2006). *Cryptography and network security principles and practices*. USA: Prentice Hall.

[5] Syam Akhil Repalle, Venkata Ratnam Kolluru (2017). *Intrusion Detection System using AI and Machine Learning Algorithm*. Koneru Lakshmaiah Educational Foundation, Andhra Pradesh, India.

[6] Mohammad Almseidin, Maen Alzubi, Szilveszter Kovacs, Mouhammad Alkasassbeh (2017). *Evaluation of Machine Learning Algorithms for Intrusion Detection System*. Mutah University, Amman, Jordan.

[7] Tom M. Mitchell. *Machine Learning* (1997). McGraw-Hill Science/Engineering/Math.

[8] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning* (2016). MIT Press. `http://www.deeplearningbook.org`.

[9] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, Emmanuel S. Pilli. *A Detailed Investigation and Analysis of using Machine Learning Techniques for Intrusion Detection*. IEEE Communications Surveys & Tutorials.

[10] Wathiq Laftah Al-Yaseen, Zulaiha Ali Othman, Mohd Zakree Ahmad Nazri. *Multi-Level Hybrid Support Vector Machine and Extreme LearningMachine Based on Modified K-means for Intrusion Detection System,Expert Systems With Applica-tions* (2016), doi: 10.1016/j.eswa.2016.09.041

[11] Anna L. Buczak, Erhan Guven. *Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection*. IEEE Communications Surveys & Tutorials, VOL. 18, NO. 2, second quarter 2016.

[12] M. Tavallaee, E. Bagheri, W. Lu, A. Ghorbani. *A Detailed Analysis of the KDD CUP 99 Data Set*, Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA), 2009.

[13] J. McHugh. *Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory.* ACM Transactions on Information and System Security, vol. 3, no. 4, pp. 262-294, 2000.

[14] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, Andreas Hotho. *A Survey of Network-based Intrusion Detection Data Sets.* ArXiv:1903.02460v2 [cs.CR]. Computer & Security 2019.

[15] Gozde Karatas, Onder Demir, Ozgur Koray Sahingoz. *Increasing the Performance of MachineLearning-Based IDSs on an Imbalancedand Up-to-Date Dataset.* IEEE Access Volume 8. 2020.

[16] Shai Shalev-Shwartz, Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithm.* Cambridge University Press. 2014.

[17] L.P. Dias, J. J. F. Cerqueira, K. D. R. Assis, R. C. Almeida Jr. *Using artificial neural network in intrusion detection systems to computer networks.* 2017 9th Computer Science and Electronic Engineering (CEEC).

[18] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin. *A Practical Guide to Support Vector Classification.* Department of Computer ScienceNational Taiwan University, Taipei 106, Taiwan. Initial version: 2003. Last updated: April 15, 2010.

[19] Wun-Hwa Chen, Sheng-Hsun Hsu, Hwang-Pin Shen. *Application of SVM and ANN for intrusion detection.* W.-H. Chen et al. / Computers & Operations Research 32 (2005) 2617 – 2634.

[20] L. Dhanabal, Dr. S.P. Shantharajah. *A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms.* International Journal of Advanced Research in Computer and Communication EngineeringVol. 4, Issue 6, June 2015