INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SECCIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

ZACATENCO

Análisis de la exactitud de los sistemas de red de detección de intrusos basados en anomalías

TESIS

QUE PARA OBTENER EL GRADO DE

MAESTRO EN CIENCIAS EN INGENIERÍA DE TELECOMUNICACIONES

PRESENTA

Óscar Azeem Becerril Domínguez

DIRECTOR DE TESIS

Dr. Felipe Rolando Menchaca García

Ciudad de México, Mayo 2018

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México siendo las 14.00 horas del día 30 del mes de Abril del 2018 se reunieron los miembros de la Comisión Revisora de Tesis designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de E.S.I.M.E. para examinar la tesis de titulada:

"ÁNALISIS DE LA EXACTITUD DE LOS SISTEMAS DE RED DE DETECCIÓN DE INSTRUSOS BASADOS EN ANOMALÍAS"

Presentada por el alumno:

BECERRIL	DOMÍNGUEZ	ÓSCAR AZEEM
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

A	1	6	1	1	4	1

aspirante de:

MAESTRO EN CIENCIAS EN INGENIERÍA DE TELECOMUNICACIONES

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***SU APROBACIÓN DE LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director de tesis

DR. FELIPE ROLANDO MENCHACA GARCÍA

Presidente

DR. MARCO ANTONIO ACEVEDO MOSQUEDA

Segundo Vocal

M. EN C. SERGIO VIDAL BELTRÁN

Tercer Vocal

DR. HÉCTOR OVIEDO GALDEANO

Secretario

M. EN C. MIGUEL SÁNCHEZ MERAZ

EL PRESIDENTE DEL COLEGIO

DR. MIGUEL TOLEDO VELÁZQUEZ

SECCIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESION DE DERECHOS

En la Ciudad de México el día 24 del mes Mayo del año 2018 , el (la) que suscribe Óscar Azeem Becerril Domínguez alumno (a) del Programa de Maestría en Ciencias en Ingeniería de Telecomunicaciones con número de registro A161141 , adscrito a S.E.P.I. E.S.I.M.E. Unidad Zacatenco, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección de Dr. Felipe Rolando Menchaca Gacía y cede los derechos del trabajo intitulado Análisis de la exactitud de los sistemas de red de detección de intrusos basados en anomalías, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección oscar.azeem@me.com Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Óscar Azeem Becerril Domínguez

Nombre y firma

Resumen

Desocupado lector, sin juramento me podrás creer que quisiera que este libro [...] fuera el más hermoso, el más gallardo y más discreto que pudiera imaginarse.

Miguel de Cervantes, Don Quijote de la Mancha

Éste trabajo de Tesis presenta un análisis de la exactitud de los sistemas de detección de intrusos en red basados en anomalías haciendo uso de la base de datos NSL-KDD99 la cual es una mejora de la base de datos KDD99 perteneciente a siete semanas del tráfico de red de una red militar con el objeto del entrenamiento y evaluación de clasificadores pertenecientes a la rama de algoritmos de aprendizaje de máquina. Al realizar el análisis de exactitud se hace uso de diferentes métricas, las cuales son específicas para la clasificación binaria y en cinco clases, debido a que el promedio de aciertos en relación a la totalidad de instancias en la base de datos se ha demostrado que no describe completamente el desempeño de los clasificadores.

Para el análisis de exactitud se evalúan además dos sistemas de red de detección de intrusos ligero con atributos reducidos, menores a los 41 atributos originales en la base de datos NSL-KDD99. Los atributos reducidos se encuentran haciendo uso del algoritmo de búsqueda heurística de trayectoria simple: Ascenso de Colinas Por Mutación Aleatoria, y del algoritmo de búsqueda heurística poblacional: Optimización por Movimiento de Iones, el cual es modificado para ser utilizado en problemas binarios y aplicado por primera vez en los sistemas de red de detección de intrusos.

Con el propósito de utilizar un sistema de red de detección de intrusos ligero con atributos reducidos, se busca determinar aquel clasificador que no esté sobre entrenado ni poco entrenado que proporcione la mayor exactitud con una menor cantidad de dimensiones y por lo tanto otorgue un menor tiempo de entrenamiento y evaluación, factor importante en la implementación en tiempo real de los sistemas de detección de intrusos.

Abstract

This Thesis introduce a network intrusion detection system accuracy analysis based on anomalies, using the NSL-KDD99 database which it's an improvement of the KDD99 database belonging to seven weeks of a militar network traffic with the porpouse of training and testing classifiers pertaining to the branch of machine learning algorithms. By doing an accuracy analysis different metrics are used, which are specific for the binary and five class classification, due to the relationship between correct classifications and the totality of instances has been proved to no describe completely the classiffiers performance.

For the accuracy analysis are evaluated furthermore two lightweight network intrusion detection systems with reduced features, less than the 41 original features in the NSL-KDD99 database. The reduced features are found by using heuristic search algorithms, Random Hill Mutation Climbing as a simple trayectory algorithm, and Ions Motion Optimization as a population algorithm. Ions Motion Optimization it's also modified to be used on binary problems and applied for the very first time on the network intrusions detection systems.

In order to use a lightweight network intrusion detection system we aim to determine the classifier that is not overfit nor underfit, which grant the biggest accuracy score with a lower quantity of features than the original NSL-KDD99, and therefore provide a lower training and testig time, important factor in a network intrusion detection system implementation in real-time.

Agradecimientos

[...] a vos
porque es una constancia
de que existís
enhorabuena,
y a nosotros también
porque es un signo
de que estamos
o estuvimos
aquí [...]

Mario Benedetti
Carta a un Joven Poeta.

Dar las gracias al Consejo Nacional de Ciencia y Tecnología, al Instituto Politécnico Nacional y, sobre todo, a la Sección de Estudios de Posgrado e Investigación de la Escuela Superior de Ingeniería Mecánica y Eléctrica unidad Zacatenco, la cual me albergo con cariño en mi tiempo de estancia.

A los profesores de los cuales tuve la fortuna de recibir clases. Al Dr. Felipe Rolando Menchaca García por brindarme todas las libertades junto con el apoyo desde que éste trabajo era proyecto y hoy es certeza. A la Dra. Yenny Villuendas Rey quién me acercó al cómputo inteligente a través de la búsqueda heurística, cuyas enseñanzas son los principios de los algoritmos aquí utilizados.

Que no todo el tiempo de instancia fue hojas sueltas y artículos, dar las gracias a compañeros que se volvieron amigos, a Tania y Samuel, con quienes el lugar de trabajo se volvió un pretexto y siempre es una maravilla compartir.

A mí Familia porque soy quien soy gracias a ella. A mi madre Consuelo Domínguez, por ser lo mejor de la vida además de mi guía. A mi padre Óscar Becerril, por las pláticas en los días de bosque. A mi hermana Daniela, porque desde ser base apenas nos separamos. A Ale, por sus brazos.

Nomenclatura

Base de datos con puntos aleatorios	R
Base de datos de entrenamiento	X
Base de datos de evaluación	χ
Clase predicha	Y
Coeficientes del hiperplano	β
Conjunto de vecinos en un punto de evaluación	η_o
Constante de Boltzmann	k_b
Desviación estándar	σ
Dimensiones a mutar entre dos vectores	D_{isp}
Entropía	$H(x)$
Factor de peso entre la tasa de error y reducción de dimensiones	α
Fuerza de atracción	f_u
Función fitness	f_{it}
Grado de pureza	G
Grado de pureza mediante entropía	GH
Información	$I(x)$
Iteraciones restantes	I
Modelo del sistema	$f(x)$
Máxima profundidad del árbol	M_p
Máximo número de dimensiones posibles a mutar	D_{max}
Población binaria de iones	E
Población de aniones	A
Población de cationes	C
Probabilidad	$P(x)$
Probabilidad condicional	P_r
Registro de conexión de evaluación	χ_i
Registro de conexión de entrenamiento	x_i
Rendimiento del clasificador	R_e
Rendimiento mediante reducción de dimensiones	F
Temperatura por iteración	T_t
Total de dimensiones	p
Total de dimensiones a mutar por iteración	N
Total de dimensiones en NSL-KDD99	T
Total de individuos en la población de iones	P_{ob}

Total de registros de conexión (instancias)	n
Total de árboles	C_a
Total posible del espacio de estados	E_p
Transformación del plano	ϕ
Vecinos a evaluar	k

Glosario

- **ARFF (Attribute Relation File Format)** Formato de texto ASCII el cual describe una lista de instancias las cuales comparten un conjunto de atributos; es usado especialmente por el software de aprendizaje de máquina WEKA.
- **Aprendizaje de máquina** Subcampo de las ciencias de la computación y una rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan a las computadoras aprender.
- **Búsqueda** Proceso de evaluar las distintas secuencias de acciones en un espacio de estados para encontrar aquellas que llevan de un estado inicial a un estado final (objetivo)
- **Búsqueda heurística** Búsqueda guiada encargada de reducir la cantidad de posibles soluciones en el espacio de estados mediante alguna analogía.
- **CSV (Comma Separated Values)** Formato de texto plano utilizado para describir listas, dónde cada símbolo "," es utilizado para denotar cada una de las características de dicha lista.
- **Clasificador** Algoritmo de Aprendizaje de máquina utilizado en la categoría de aprendizaje supervisado. Se encarga de predecir la categoría de alguna instancia a partir de clases previamente definidas.
- **Curva ROC (Receiver Operating Characteristic)** A través de los resultados Verdaderos Positivos (VP) y Falsos Positivos (FP), la curva ROC se define mediante FP en el eje x y VP en el eje y, dónde cada punto representa un diferente ejercicio de evaluación de algún algoritmo de aprendizaje de máquina. Da cómo resultado una representación gráfica de la sensibilidad (VP) frente a la especificidad (FP).
- **DoS** Tipo de ataque en el cual un atacante envía de forma continua falsas solicitudes a un servidor, agotando los recursos de memoria, y por lo tanto el servidor comienza a rechazar solicitudes legitimas de los usuarios.

- **Espacio de Estados** Total de posibles soluciones para algún problema específico de optimización (minimización/maximización).
- **HIDS (Host Intrusion Detection System)** Sistema de detección de intrusos basado en Host. Su representación más conocida son los sistemas antivirus.
- **IDS (Intrusion Detection System)** Sistema de detección de intrusos. Su representación más conocida son los sistemas NIDS basados en anomalías.
- **Instancia** Elemento particular de alguna base de datos. Su formato se muestra debido al tipo de archivo en el que se encuentre, CSV, ARFF, etc.
- **Matriz de Confusión** También llamada matriz de errores, es una tabla la cual permite observar de forma detallada el rendimiento de algún algoritmo de aprendizaje de máquina. Su uso se da en situaciones dónde se realiza un aprendizaje supervisado.
- **Minado de datos** Es el proceso de descubrir patrones en grandes bases de datos mediante métodos estadísticos y de aprendizaje de máquina.
- **NIDS (Network Intrusion Detection System)** Sistema de detección de intrusos basado en Red. Se encarga de crear un perfil del tráfico normal de la red de datos utilizando técnicas de aprendizaje de máquina, con las cuales puede discernir si una nueva conexión es considerada cómo maliciosa.
- **Payload** Conjunto de datos pertenecientes únicamente a la información del mensaje, excluyendo cabeceras.
- **Probe** Tipo de ataque en el cual un atacante intenta conseguir información sobre la taxonomía de la red de computadoras que desea vulnerar con el propósito de eludir los controles de seguridad.
- **R2L** Tipo de ataque en el cual un atacante envía paquetes a una máquina a través de la red sin que él posea una cuenta en dicha máquina, tratando de explotar alguna vulnerabilidad con el objetivo de ganar acceso a ella.
- **Score** Tipo de métrica utilizada para evaluar la exactitud de un clasificador. Es la relación de instancias correctamente clasificadas y el total de instancias en una base de datos de evaluación.
- **U2R** Tipo de ataque en el cual un atacante tiene acceso local a través de una cuenta y trata de ganar privilegios de administrador.

Índice

Índice de figuras

Índice de Tablas

Índice de Algoritmos

Capítulo 1

Introducción

Púsose don Quijote delante de dicho carro, y haciendo en su fantasía uno de los más desvariados discursos que jamás había hecho, dijo en alta voz:

Alonso Fernández de Avellaneda, El Ingenioso Hidalgo Don Quijote de la Mancha

Resumen: El presente capítulo describe primero un panorama general de los sistemas de detección de intrusos, en el cual se muestra su clasificación haciendo énfasis en los sistemas de detección de intrusos en red basados en anomalías y su utilidad para detectar nuevas amenazas. Posteriormente de describe el planteamiento de solución de éste trabajo de Tesis al problema de exactitud y alto tiempo de entrenamiento de los clasificadores en los algoritmos de aprendizaje de maquina al ser aplicados en los sistemas de detección de intrusos. Por último se presentan la justificación del trabajo, la hipótesis a partir de la cual se plantea su solución, el objetivo general, los objetivos específicos, los alcances del trabajo y el contenido restante de la tesis.

1.1. Sistemas de Detección de intrusos

El propósito de un sistema de detección de intrusos (IDS) es detectar algún acceso no autorizado. Los dispositivos IDS pueden ser ampliamente categorizados de acuerdo a las técnicas de detección utilizadas (detección de anomalías, examinación de paquetes, o basados en reglas). El concepto de la *detección de intrusos* comenzó en la década de 1980. Unos de los trabajos más tempranos en la detección de intrusos fue realizada en una agencia de gobierno por Jim Anderson [1], uno de los fundadores de los sistemas IDS,

concentrandose en maneras de mejorar la seguridad y los sistemas de vigilancia. En su trabajo Anderson propone maneras de revisar minuciosamente los logs de auditorias para la detección de intrusos. Realizó una comparación de éstos cuando usuarios de la red querían vulnerar archivos o computadoras, comparándolos con aquellos de uso normal. Es importante destacar que éste primer trabajo estaba unicamente enfocado en hosts. Después Like Anderson, y Denning [2] se enfocaron en procesar logs de auditoria para violaciones de seguridad mediante el escaneo de cualquier uso anormal. Ésta fue la base para que Lunt y Jagannathan en [3] usaran éste enfoque para crear un sistema IDS basado en host (HIDS) el cual determinara el comportamiento normal de logs de auditoria anteriores antes de aplicarlo a logs de auditoria actuales para detectar e identificar potenciales intrusiones.

En la década de los años 90's Heberlin junto con su equipo [4] extendieron el IDS basado en host propuesto por Denning para incluir el *monitoreo de red*. El IDS propuesto analiza el tráfico de la red y compara comportamientos actuales y anteriores para descubrir anomalías, y por lo tanto, intrusos, éste sistema se denomina sistema de detección de intrusos basado en red (NIDS). En 1998 la Agencia de proyectos de investigación de defensa Avanzada (DARPA) evalúo los sistemas IDS de la época. Incluso en los mejores sistemas la taza de detección era considerada demasiada baja, especialmente para la detección de nuevos ataques [5]. Lee and Zhang [6] señalaron lo que sería la base en la aplicación de cualquier sistema IDS, *un sistema IDS debe estar estrategicamente colocado en switches, routers o gateways para recoger información de los puntos de trafico visible*. En 1999 surge Knowledge Discovery Database 99 (KDD99) [7], la primer base de datos utilizada cómo punto de referencia para el entrenamiento y evaluación de los sistemas NIDS basados en anomalías perteneciente al almacenamiento de siete semanas del tráfico de red de una red militar, resumido en conexiones de red con 41 atributos por conexión, divididos en 5 clases. J. McHugh [8] expone en el 2000 los problemas inherentes en la elaboración y distribución de las 5 clases en la base de datos KDD99, así cómo en el volcado del tráfico mediante TCPDUMP en [7].
En 2009 Mahbod y Ebrahim [9] realizan un estudio completo de la distribución de ataques y la exactitud de diversos sistemas de red de detección de intrusos los cuales son entrenados haciendo uso de KDD99, demostrando una distribución focalizada en ciertas clases y múltiples instancias redundantes. En [9] se propone el uso de una nueva base de datos para el entrenamiento de los sistemas de red de detección de intrusos NSL-KDD la cual soluciona la mayoría de los problemas inherentes en KDD99. La base de datos NSL-KDD se ha convertido en el nuevo punto de referencia para el entrenamiento y evaluación de los sistemas NIDS, si bien aún sufre de algunos problemas mostrados en [8] y podrá no ser una perfecta representación de las redes de datos existentes en la actualidad, debido a la falta de bases de datos pu-

blicas existentes y a trabajos contemporáneos los cuales hacen uso de ella, éste trabajo considera que aún puede ser aplicada y otorgar un método de evaluación efectivo para realizar un análisis de la exactitud los diferentes sistemas de red de detección de intrusos basados en anomalías.

1.2. Algoritmos de Aprendizaje de Máquina

Los sistemas de red de detección de intrusos hacen uso de una base de datos de entrenamiento para ser utilizada en los algoritmos de aprendizaje de máquina y de una base de datos de evaluación con la cual es posible tener una métrica de su exactitud. Los algoritmos de aprendizaje de máquina usados en los sistemas NIDS son clasificadores. Del año 2000 hasta el año 2009 se han realizado varios experimentos de diferentes clasificadores usando la base de datos KDD99. En 2002 Maheshkumar Sabhnani [10] propone un analisis de los clasificadores, Multilayer Perceptron (MLP), Gaussian Classifier (GC), Decision Trees [11] haciendo uso del software LNKnet, obteniendo un resultado del 97 % para la clasificación del tipo DoS, resultado no conseguible en un escenario real y pese a no realizar un preprocesado en la base de datos KDD99, demostrando los problemas descritos por J. McHugh en [8]. En el 2014 Muhamad y Dewan [12] realizan un estudio del clasificador Support Vector Machines (SVMs), haciendo uso de la base de datos NSL-KDD para el entrenamiento y evaluación de un sistema NIDS, en el cual obtienen un 82.37 % de exactitud usando 41 atributos, paralelamente obtienen un 78.85 % en la clasificación usando solamente 3 atributos, empero a, no hacen un estudio de los tiempos de entrenamiento y evaluación del clasificador SVMs; sus resultados muestran valores aproximados a los obtenidos en un sistema NIDS funcionando en tiempo real, en contraste con [10].
En tiempos recientes se han utilizado algoritmos de reducción de dimensiones para disminuir el total de 41 atributos en la base de datos NSL-KDD, con el fin de disminuir los tiempos de entrenamiento y evaluación de los clasificadores en los sistemas NIDS. Métodos del tipo filtro y envoltura han sido implementados con éxito. Dentro de los métodos del tipo envoltura la búsqueda heurística es ampliamente utilizada debido a versatilidad de sus algoritmos, un mejor resultado en la exactitud de los sistemas NIDS con características reducidas en contraste con los métodos del tipo filtro, y pese a requerir un mayor tiempo de cómputo al tener los algoritmos de aprendizaje de máquina embebidos en la función de evaluación de la búsqueda heurística. En [13] You Chen y Wen-Fa Li, realizan un sistema NIDS ligero con caracteristicas reducidas haciendo uso de dos algoritmos de búsqueda heuristica: Random Hill Mutation Climbing (RMHC) y Simulated Annealing (SA); en éste estudio se logra una reducción de hasta el 50 % en los tiempos de evaluación y entrenamiento de los algoritmos de aprendizaje de máquina en los sistemas NIDS, haciendo uso de la base de datos KDD99. Bajo el plantea-

miento de la búsqueda heurística se ha demostrado que es posible reducir el tiempo de evaluación y entrenamiento de los algoritmos de aprendizaje de máquina en los sistemas NIDS, sin afectar de forma drástica su exactitud.

1.3. Planteamiento del problema

En 1969 surge la red, Advanced Research Projects Agency Network (ARPANET), la cual fue la primer red en implementar el conjunto de protocolos TCP/IP, cuyas tecnologías se convirtieron en la fundación técnica de lo que hoy conocemos cómo Internet. En los origenes de ARPANET *la seguridad era mínima*. Internet al basarse en la idea de múltiples redes independientes, ha crecido a escalas impensadas y con ello los protocolos de seguridad y amenazas a las redes de datos [14]. En tiempos actuales la banca en línea ha mantenido una posición creciente y tomado una porción más grande del mercado economico, los hospitales han migrado los historiales de sus pacientes a bases de datos. La seguridad en éstas entidades tiene una importancia crítica y siempre debe de asegurarse.

Existen métodos estáticos para asegurar la confidencialidad, integridad y disponibilidad de un sistema o red, entre los cuales se encuentran las políticas de seguridad, el cifrado, control de accesos, sistemas de Firewall, sistemas de detección de intrusos basados en el mal uso, entre otros. Todos los anteriores tienen la característica de necesitar asistencia en el momento de su despliegue sumado a la incapacidad para detectar amenazas nuevas, las cuales no hayan sido previamente contempladas y/o registradas en alguna base de datos [15]

Los sistemas de detección de intrusos en red basados en anomalías solucionan las problemáticas de los sistemas estáticos presentados en [15], poseen la capacidad de aprender del comportamiento normal del flujo de datos en una red y en consecuencia detectar ataques a la red que no hayan sido previamente registrados. Sin embargo, los sistemas NIDS sufren de altas tazas de falsos positivos/negativos lo que dificulta su implementación.

Las soluciones desarrolladas hasta la actualidad al problema del análisis de la exactitud de los clasificadores aplicados en los sistemas NIDS, no usan una métrica común al momento de utilizar una base de datos para el entrenamiento y evaluación de los clasificadores, de lo antedicho se desprende que no es posible realizar una comparativa adecuada. En [15] [16] [17] hacen uso de la base de datos KDD99, aún cuando no se recomienda su uso. Mientras que en [18] los autores fusionan la base de datos de entrenamiento junto con la base de datos de evaluación, para después tomar porciones aleatorias de ella para formar su propia base de datos de entrenamiento y evaluación, en consecuencia es imposible obtener una comparación valida al no poder generar las mismas bases de datos con los cuales entrenar y evaluar los clasificadores. En [19] se realiza el análisis del desempeño del clasificador SVMs haciendo uso de la base de datos NSL-KDD, utilizando un sistema NIDS con 41 atribu-

tos y otro con atributos reducidos el cual es generado empleando algoritmos genéticos, no obstante, además de no mostrar ninguna configuración propia de los algoritmos genéticos cómo el método de cruzamiento, la selección de padres, o el método de mutación así como su probabilidad, no es realizado ningún estudio con otro clasificador ni haciendo uso de ninguna otra heurística con la cual evaluar su exactitud. Al mismo tiempo en [13] [20] [21] usando la base de datos NSL-KDD realizan el estudio de la exactitud de un sistema NIDS usando únicamente un clasificador y sin considerar la evaluación del tiempo de entrenamiento. En [22] realizan un estudio de los sistemas NIDS usando diferentes clasificadores entrenados con la base de datos NSL-KDD, haciendo uso del software comercial WEKA [23], mostrando únicamente su desempeño sin especificar parámetros específicos de cada uno de los clasificadores. En éste trabajo de tesis se da una solución al problema de la falta de un análisis exhaustivo en el análisis de la exactitud de los clasificadores en los algoritmos de aprendizaje de máquina aplicados en los sistemas NIDS, utilizando un sistema con 41 dimensiones y dos sistemas con dimensiones reducidas.

1.4. Propuesta de solución

En éste trabajo de tesis se propone realizar una comparativa del análisis de exactitud de los algoritmos clasificadores K-Nearest Neighbors (KNN), Decision Trees (DT), Random Forest (RF) y Support Vector Machines (SVMs) aplicados en los sistemas NIDS, siendo entrenados mediante la base de datos NSL-KDD la cual es una mejora de KDD99 y soluciona algunos de sus problemas inherentes. Se establece que algoritmo de aprendizaje de máquina es más ventajoso frente a otros, siendo evaluados mediante el acierto o error en su clasificación, además del tiempo de su construcción. Los clasificadores son aplicados para detectar anomalías en el caso de dos clases (clasificador binario) ataque no ataque, y en el caso de cinco clases, ataques del tipo Denial-of-Service (DoS), User to Root (U2R), Remote to Local (R2L) y Probing (Probe).

Las métricas de evaluación en la exactitud de los clasificadores incluyen además del porcentaje de aciertos y errores en la clasificación (Score), la curva Receiver Operating Characteristic (ROC) para el caso de la clasificación binaria.

El criterio que se ha usado para utilizar los clasificadores anteriormente dichos, es el buen desempeño que han mostrado en trabajos similares [10] [21].

Para el análisis de la exactitud de los clasificadores se ha hecho uso al mismo tiempo de Reducción de Dimensiones (RD), el cual es un método para la selección de dimensiones obteniendo un conjunto de tamaño menor al conjunto de 41 dimensiones original propio de NSL-KDD y, en consecuencia, es posible realizar una comparativa entre la exactitud y tiempos de computo

de los clasificadores en un sistema NIDS con 41 dimensiones y los dos sistemas con dimensiones reducidas.

Así, en efecto, la reducción de dimensiones permite mejorar el tiempo de entrenamiento de los algoritmos de aprendizaje de máquina, evitando reducir de forma drástica la exactitud de su clasificación.

Para la reducción de dimensiones se ha utilizado el método del tipo envoltura, a través de la búsqueda heurística utilizando los algoritmos de trayectoria simple, Random Mutation Hill Climbing (RMHC), Simulated Annealing (SA) y el algoritmo del tipo poblacional Ions Motion Algorithm (IMO); dónde IMO cuenta con una población de 20 individuos y cuyas características propias se describen más adelante.

1.5. Justificación

Los sistemas de red de detección de intrusos basados en anomalías tienen la capacidad de prevenir algún acceso no autorizado a la red de datos. Son una solución dinámica con la capacidad de aprender del tráfico normal de la red y considerar algún patrón que se desvié de éste cómo anomalía (ataque) y al mismo tiempo clasificarla, en contraste con los sistemas de seguridad estáticos los cuales no son capaces de detectar algún ataque a la red del cual no se tenga conocimiento previo.

El tiempo de computo de los algoritmos de aprendizaje de máquina utilizados en los sistemas NIDS está directamente relacionado a la cantidad de dimensiones en la base de datos con la cual son entrenados, 41 dimensiones para el caso de la base de datos NSL-KDD utilizada en éste trabajo de tesis. Reducir su cantidad de dimensiones no de forma arbitraria, sino al encontrar aquellas que reducen de forma mínima la exactitud de los clasificadores, propicia que se minimice su tiempo de computo requerido con el objeto de sentar las bases para su aplicación en tiempo real.

El tener un análisis exhaustivo de los algoritmos de aprendizaje de máquina aplicados en los sistemas NIDS otorga aquellos clasificadores, parámetros y dimensiones que poseen un mayor impacto en futuras implementaciones.

1.6. Hipótesis

Al realizar un análisis de los clasificadores en los sistemas NIDS obteniendo la mayor exactitud y el menor tiempo de computo, es posible obtener aquellos parámetros que proporcionen mejores resultados los cuales puedan ser implementados en un escenario dónde se despliegue un NIDS en una red de datos real.

1.7. Objetivo General

Realizar un análisis de la exactitud de los algoritmos de aprendizaje de máquina aplicados en los sistemas de red de detección de intrusos haciendo uso de la base de datos NSL-KDD con 41 dimensiones y dimensiones reducidas utilizando algoritmos de búsqueda heurística.

1.8. Objetivos Específicos

- Realizar un estudio del estado del arte de los sistemas de detección de intrusos basados en host y en red, de los clasificadores en los algoritmos de aprendizaje de máquina, y de los métodos de búsqueda heuristica utilizados con el fin de reducir dimensiones.
- Realizar un preprocesado de la base de datos NSL-KDD la cual es usada para el entrenamiento y evaluación de todos los clasificadores en éste trabajo de tesis, en términos de normalización y ganancia de la información de cada una de sus dimensiones.
- Realizar un estudio del análisis de la exactitud de diversos clasificadores aplicados en los sistemas NIDS para el caso de 2 y 5 clases utilizando para su evaluación múltiples métricas.
- Realizar un estudio del análisis de la exactitud de diversos clasificadores aplicados en los sistemas NIDS en el caso de 41 dimensiones y con dimensiones reducidas utilizando algoritmos de búsqueda heurística.
- Comprobar los resultados de la exactitud de los diferentes clasificadores con aquellos analisis existentes en la literatura que hagan uso de la base de datos NSL-KDD.

1.9. Alcances del Trabajo

Dentro de los sistemas de detección de intrusos existen dos vertientes, aquellos basados en host y en red. En éste trabajo se hace uso de los sistemas de detección de intrusos basados en red debido a su capacidad de detectar nuevas amenazas. En los algoritmos de aprendizaje de máquina existen aquellos enfocados en el aprendizaje supervisado y no supervisado, en éste trabajo de tesis se hace uso de los algoritmos de aprendizaje de máquina de aprendizaje supervisado al hacer uso de la base de datos NSL-KDD dónde cada instancia se encuentra correctamente etiquetada con un valor correspondiente a las clases Normal, U2R, R2L, Probe y DoS. Por lo antedicho no se hace uso de los algoritmos de aprendizaje de maquina sin supervisar, cómo los del tipo cluster.

Los algoritmos de aprendizaje de máquina utilizan la base de datos NSL-KDD para su entrenamiento y evaluación, sin embargo NSL-KDD aún cuenta con algunos problemas propios de la base de datos KDD99 de la cual está basada y puede no representar de forma adecuada las redes actuales, pese a ésto, debido a la falta de bases de datos públicas representativas del tráfico de una red de datos, a la dificil generación de una base de datos particular, y al uso en la actualidad de la base de datos NSL-KDD en trabajos similares, se considera que aún puede ser ocupada y otorgar un análisis confiable del desempeño de diversos clasificadores en sistemas NIDS. Trabajos similares hacen uso, en el estado del arte y en la actualidad, de la base de datos KDD99, sin embargo éste trabajo hace uso únicamente de la base de datos NSL-KDD debido a que en [8] se mostraron algunas de sus deficiencias y no es recomendado su uso.

1.10. Contenido Restante de la Tesis

En el capítulo dos se presenta el estado del arte de la seguridad de la información. Detalla el surgimiento y evolución de los sistemas de red de detección de intrusos (IDS) a aquellos basados en anomalías. Defiende y justifica el uso de la base de datos NSL-KDD99 para el entrenamiento y evaluación de los sistemas IDS. Muestra la taxonomía de los algoritmos de aprendizaje de máquina con el objeto de profundizar en aquellos algoritmos no paramétricos utilizados en éste trabajo de tesis. Ahonda y muestra las bases de los métodos de reducción de dimensiones, siendo específicos para el caso de la búsqueda heurística.

En el capítulo tres se describe la propuesta de solución de este trabajo de Tesis al problema de la carencia de un análisis exhaustivo y replicable de los sistemas de red de detección de intrusos basados en anomalías utilizando algoritmos de aprendizaje de máquina para su evaluación mediante un sistema de dimensiones completas (utilizando las 40 dimensiones de NSLKDD y ENSLKDD) y dos sistemas de dimensiones reducidas obtenidas mediante búsqueda heurística. Se muestra el proceso de búsqueda heurística modificado con el objeto de reducir explícitamente las dimensiones en el sistema IDS utilizando el algoritmo RMHC junto con SA, además se detalla el proceso de búsqueda heurística IMO modificado de manera original aplicado al problema binario que representan los sistemas IDS, con el objetivo explícito de reducir dimensiones.

En el capítulo 4 se muestran los experimentos realizados y los resultados obtenidos en los sistemas IDS con dimensiones completas y los dos sistemas con dimensiones reducidas obtenidos mediante la búsqueda heurística. Se describe el entorno en el que se desarrollan los algoritmos de aprendizaje de máquina, se mencionan las librerías utilizadas y el código en lenguaje Python implementado para la búsqueda heurística. Se evalúan los algoritmos KNN,

SVM's, DT y RF, en el caso de 2 y 5 clases, para determinar el clasificador que muestre el mejor equilibrio entre porcentaje de aciertos y tiempo de entrenamiento/evaluación. Se realiza una comparativa en términos de porcentaje de exactitud y tiempo de entrenamiento/evaluación del clasificador para un sistema con 40 dimensiones y dimensiones reducidas.

En el capítulo cinco se presentan las conclusiones obtenidas y los trabajos futuros. Se presenta la bibliografía consultada en forma detallada. Por último se muestran en los apéndices los algoritmos utilizados para el preprocesado de la base de datos, así como el utilizado para su visualización, los códigos de búsqueda heurística implementados en los dos sistemas IDS con dimensiones reducidas, además de los resúmenes de ponencias en congresos y coloquios, desprendidos de éste trabajo.

Capítulo 2

Estado del arte

Obra de modo que merezcas a tu propio juicio y a juicio de los demás la eternidad, que te hagas insustituible, que no merezcas morir.

Miguel de Unamuno,
El Problema Práctico

Resumen: El presente capítulo muestra el estado del arte de la seguridad de la información. Detalla el surgimiento y evolución de los sistemas de red de detección de intrusos (IDS) a aquellos basados en anomalías. Realiza una evaluación de los trabajos contemporáneos en los sistemas IDS que tienen el objeto de analizar la exactitud de los algoritmos de aprendizaje de máquina y resalta sus carencias además de la imposibilidad de réplica. Defiende y justifica el uso de la base de datos NSL-KDD99 para el entrenamiento y evaluación de los sistemas IDS. Muestra la taxonomía de los algoritmos de aprendizaje de máquina con el objeto de profundizar en aquellos algoritmos no paramétricos utilizados en éste trabajo de tesis. Por último ahonda en los métodos de reducción de dimensiones con el objeto de optimizar el tiempo de evaluación de los algoritmos de aprendizaje de máquina siendo justificada la implementación del método de búsqueda heurística aplicado en éste trabajo.

2.1. Seguridad de la información

La seguridad de la información, llamada también InfoSec, se define cómo: "Preservación de confidencialidad, integridad y disponibilidad de la información. Dónde se involucran, además, propiedades cómo la autenticidad,

responsabilidad, y confiabilidad." [24]. La seguridad de la información se relaciona con la implementación de diversos mecanismos de seguridad tales cómo técnicos, organizacionales, legales y humanos, todo con el fin de mantener la información libre de toda amenaza. El área de principal importancia para el campo de la seguridad de la información, es el tener una protección balanceada de la Confidencialidad, Integridad y Disponibilidad de los datos, también conocida cómo la triada CIA, mientras se mantienen los objetivos en la implementación de políticas eficientes, reduciendo en lo posible, la productividad de la organización [25]. La triada CIA se describe a continuación y la Figura 2.1 muestra su representación gráfica.

- **Confidencialidad de la información.** La confidencialidad de la información, en términos generales es el proporcionar libertad al usuario de cualquier interferencia externa. Es la habilidad de proporcionar la confianza a los usuarios sobre la privacidad de información sensible mediante diversos mecanismos de tal manera que su acceso pueda ser realizado únicamente por usuarios permitidos. Los mecanismos para proporcionar confidencialidad de la información, incluyen entre otros, encriptación, verificación en dos pasos, verificación biométrica.

- **Integridad de la información.** Durante el proceso de comunicación entre dos terminales, la información puede ser alterada por cibercriminales o incluso por factores inherentes en aparatos tecnologicos cómo el fallo en los métodos de almacenamiento. La integridad de la información se refiere a la protección de información util de cibercriminales o de interferencias externas durante la transmisión y recepción, de tal manera que la información no puede ser manipulada sin que el sistema detecte la amenaza. [26]

- **Disponibilidad de la información.** Uno de los máximos objetivos de la seguridad de la información es tener la información disponible a sus usuarios, siempre que se necesite. La disponibilidad de la información asegura el acceso inmediato de usuarios autorizados no solamente en condiciones de operación normales sino también en condiciones de desastre. Los métodos de redundancia y recuperación de fallos proporcionan un duplicado de los componentes del sistema en caso de algún fallo para asegurar la confiabilidad y disponibilidad de la información.

Figura 2.1: Triada de la seguridad de la información

2.1.1. Seguridad de Red

Aunado a la seguridad de la información, se encuentra la seguridad en red, la cual está enfocada en adoptar políticas y prácticas para prevenir y monitorear un acceso no autorizado, un mal uso, o la negación de servicios de red a ciertas computadoras; cubre redes de computadoras tanto públicas cómo privadas, tales cómo las redes internas de una compañía. Éstas redes de datos son objeto de ataques de fuentes maliciosas. Los ataques pueden provenir de dos categorías, Pasivas, cuando un intruso intercepta el flujo de datos a través de la red, y activa, en el cual un intruso inicia secuencias de acciones para interrumpir la operación normal de la red o realizar un reconocimiento de la topología de red con el objeto de ganar acceso a los datos que la red tenga disponible [27].

La seguridad en redes de datos empieza con la autenticación a través de un usuario y contraseña, al poseer éste método un único factor, se le denomina autenticación de un paso. La autenticación en dos pasos requiere también, algo que el usuario posea (comúnmente enviado en un dispositivo externo), por ejemplo, un token de seguridad, una llave digital o un código enviado a través del teléfono celular. La autenticación en tres pasos requiere, además, el uso literal del usuario, a través del escaneo de retina o las huellas digitales. Una vez que el usuario ha sido autenticado, un firewall impone políticas de acceso tales cómo qué servicios están disponibles para ser accesados por los usuarios de la red. Aunque a veces efectivo para prever un acceso no autorizado, el firewall suele fallar en el análisis de contenido dañino cómo gusanos, troyanos, o diferentes tipos de malware siendo transmitidos por la red. Para asegurar la seguridad en la red, los sotware antivirus o también llamados sistemas de detección de intrusos basados en host (HIDS), ayudan a detectar e inhibir un posible malware. Los sistemas de red de detección de intrusos (NIDS) se encargan de monitorear el tráfico de la red, almacenando registros para un posterior análisis; pueden detectar amenazas no conocidas.

2.2. Sistemas de detección de intrusos

El término de intrusión se refiere a cualquier acceso no autorizado (físico o software) que trate de comprometer la confidencialidad, integridad y disponibilidad de la información. De forma general cualquier uso malicioso o indebido se refiere a una intrusión, dónde el intruso trata de encontrar alguna vulnerabilidad en el sistema de seguridad y después realizar un ataque.
La detección de intrusos es el proceso de una pronta detección sobre alguna violación en el comportamiento normal del sistema debido a acciones ilegales (ataques) realizadas por un usuario malicioso (atacante) [28]. Los sistemas de detección de intrusos (IDS) se ocupan de la detección de tal tipo de ataques en tiempo real y reportan, alertan o toman contramedidas en el ataque dando aviso al administrador. Éstos sistemas IDS son software, hardware o una combinación de ambos, usados para detectar la actividad de un intruso con la ayuda de un análisis previo obtenido a través de registros del sistema y/o técnicas de inferencia de los sistemas expertos, tales cómo el aprendizaje de máquina.

Las técnicas de detección de intrusos están clasificadas cómo detección por mal uso (en inglés Misuse) y detección por anomalías. La detección por mal uso involucra identificar ataques basados en las características de ataques definidos previamente a través de los registros del sistema. Las técnicas de detección de intrusos basados en anomalías, establecen que cualquier variación en el comportamiento del comportamiento normal (del host o red) el cual es previamente estudiado y almacenado sobre un periodo de tiempo es identificado como actividad anómala. Los sistemas IDS son clasificados en algunas de las siguientes tres clases, siendo ocupado los sistemas NIDS en éste trabajo de tesis.

- IDS basados en host (HIDS): el sistema detecta ataques dirigidos a un único host, haciendo uso de los logs del sistema.
- IDS distribuidos: el sistema detecta ataques dirigidos a múltiples hosts.
- IDS basados en red (NIDS): éste sistema detecta ataques haciendo un análisis del flujo de tráfico de la red.

2.2.1. Basados el mal uso (Misuse)

Los sistemas IDS misuse pueden detectar ataques conocidos con una alta tasa de éxito y bajo costo computacional, pero se vuelven impotentes cuando se trata de detectar ataques no conocidos [29]. Son también llamados de detección basada en firmas, dónde la firma de un ataque es un arreglo único de información la cual puede ser usada para identificar el intento de un atacante de explotar un sistema operativo o la vulnerabilidad de una

aplicación, en [30] la empresa de seguridad Symantec ofrece un listado de firmas de ataques.

Los sistemas IDS misuse están basados en reglas, tanto configuradas previamente por el sistema o configuradas de forma manual por el administrador. Éstos sistemas hacen uso de dichas reglas de manera que se encuentran realizando de forma continua una búsqueda por firmas previamente almacenadas en una base de datos, realizando ésta búsqueda paralelamente dentro de la red y/o en las operaciones del sistema tratando de detectar un ataque previamente registrado el cual es considerado cómo misuse, la Figura 2.2 muestra el diagrama de flujo de su funcionamiento.

De forma muy general se puede considerar a los sistemas IDS misuse cómo un firewall basado en reglas de negación muy específicas, por ejemplo, el detectar cuando se realiza un escaneo a ciertos puertos, o haciendo un análisis de los comandos del usuario buscando algún comportamiento abusivo.

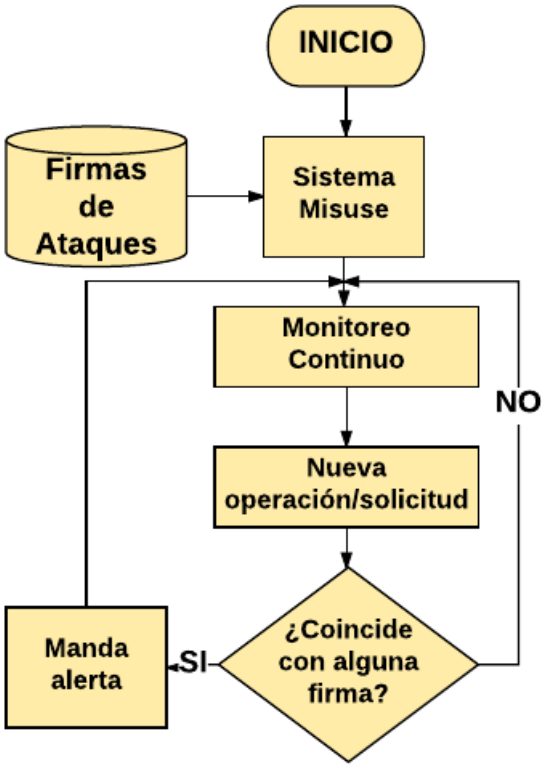


Figura 2.2: Diagrama de flujo de los sistemas IDS Misuse

Los antivirus son los sistemas HIDS misuse más representativos, al comparar firmas de posibles ataques con su base de datos. Para el caso de los

sistemas NIDS misuse, los software de código abierto Snort [31], Bro [32] y Suricata [33], son los más utilizados. Sin embargo, la limitación de los sistemas IDS misuse es que no son adecuados para detectar nuevos ataques y es común que los atacantes vislumbren nuevas formas para saltar la detección IDS [34].

2.2.2. Basados en Anomalías

Los sistemas IDS basados en anomalías encuentran ataques a través de detectar cambios en el patrón de utilización o comportamiento del sistema o red. La técnica de detección de los sistemas IDS basados en anomalías identifica aquellas actividades observadas del sistema o red que se desvían significativamente del uso normal cómo intrusiones [35]. Por consiguiente la detección mediante anomalías puede detectar intrusiones previamente *desconocidas*, las cuales no pueden ser detectadas por los sistemas misuse.

Un sistema IDS basado en anomalías tiene una fase de aprendizaje y detección. Durante la fase de aprendizaje éste aprende los perfiles del tráfico normal de la red para el caso de los sistemas NIDS, o de los perfiles en los registros del host para el caso de los sistemas HIDS y los compara con una nueva solicitud para encontrar un nivel de disimilitud, si el nivel excede un cierto nivel de umbral, un ataque es detectado. Para ambas fases se hace uso de una base de datos particular, una correspondiente a la fase de entrenamiento (aprendizaje) y otra correspondiente la fase de evaluación (detección), la Figura 2.3 describe en forma de diagrama de flujo el funcionamiento de los sistemas IDS basados en anomalías. Es posible observar que cuenta con una base de datos específica con la cual los algoritmos de aprendizaje de máquina son entrenados y a través de ellos es posible crear un perfil del tráfico de la red. Cuando se recibe una nueva solicitud se compara con éste perfil creado para discernir si existe algún tipo de anomalía.

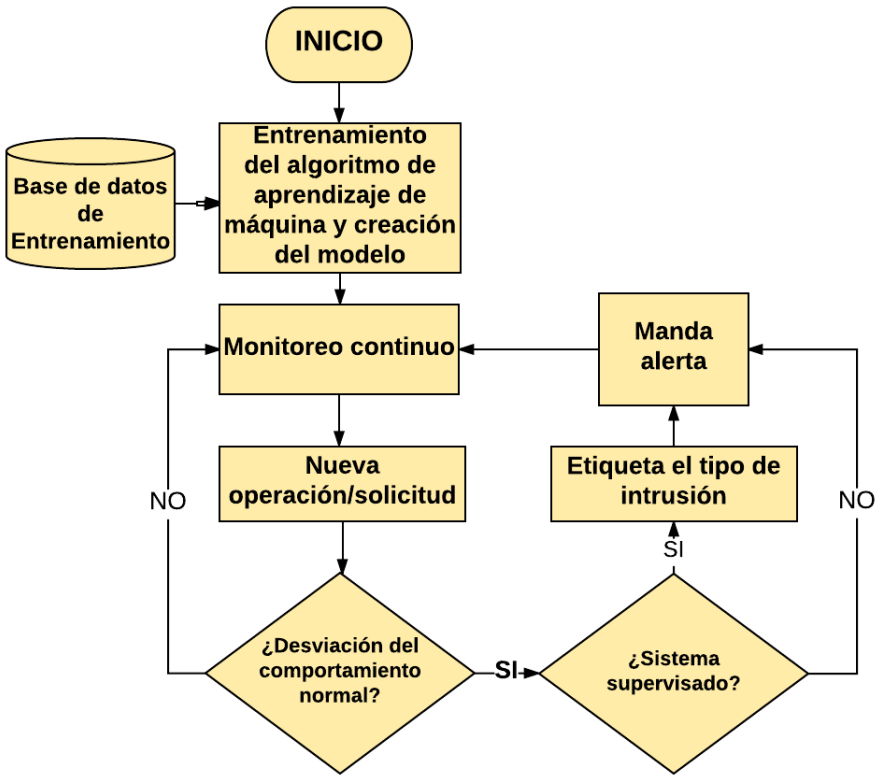


Figura 2.3: Diagrama de flujo de los sistemas IDS basados en anomalías

Los sitemas IDS basados en anomalías tienen la ventaja de detectar ataques nuevos, pero con el costo de un alto número de detecciones incorrectas, su principal problema son las alarmas falso-positivas y falso-negativas. Las alarmas falso-positivas son las actividades detectadas por el sistema IDS cómo ataques pero no lo son, mientras que las alarmas falso-negativas son las actividades que son ataques pero omitidas por los IDS. De ésta problemática surge la importancia en solucionar la enorme cantidad de alarmas falso-positivas y falso-negativas de los sistemas IDS basados en anomalías, la cual dificulta su implementación en el despliegue de una red real. Para abordar ésta limitación, los sistemas IDS basados en anomalías están teniendo mucha atención de la comunidad científica [12-15]. En consecuencia se ha empleado en tiempos recientes una técnica de computación inteligente conocida cómo aprendizaje de máquina, con la cual sea factible la implementación de los sistemas de detección de intrusos basados en anomalías. Éstas técnicas aprenden automáticamente de la información o extraen algún patrón útil de ella cómo referencia para el perfil del comportamiento del trafico normal/ataque para una subsecuente clasificación del tráfico de red.

Éste trabajo de tesis hace un análisis de la exactitud de los sistemas IDS basados en red (NIDS), basados en anomalías, por ésta razón en la fase de aprendizaje se hace uso de la base de datos NSL-KDD de entrenamiento y en la fase de detección se hace uso de la base de datos NSL-KDD de evaluación, en la sección 2.3.2 se hace un estudio detallado de ella.

2.3. Datasets para los IDS

Cómo se ha mencionado, un sistema IDS basado en anomalías tiene una fase de aprendizaje y detección, dónde cada una de las fases hace uso de su correspondiente base de datos (en inglés dataset), un dataset de entrenamiento (aprendizaje) para la creación del modelo del algoritmo de aprendizaje de máquina y un dataset de evaluación (detección) para analizar la eficacia del algoritmo de aprendizaje de máquina utilizado.

En el año de 1999 surge la base de datos que ha sido punto de referencia en el entrenamiento y evaluación de los clasificadores en los algoritmos de ML, el dataset KDD99 [7]. En su creación y hasta el momento, constituye la única base de datos original disponible de forma pública para el entrenamiento y evaluación de los sistemas NIDS de la cual el autor está enterado. En el año 2000, J. McHugh [8] realiza una fuerte crítica en los procedimientos de su creación, así cómo en la distribución de los ataques que ella contiene, sin embargo su uso se extiende hasta el año 2009 con la propuesta de Tavallaee [9] para la generación de una nueva base de datos llamada NSL-KDD. En [9] se exponen las problemáticas más representativas de KDD99 y se da una solución para la cantidad de vectores duplicados (redundantes) que ella contiene, así como la distribución de ataques focalizada en los del tipo DoS. KDD99 se muestra a detalle en la Sección 2.3.1 mientras que NSL-KDD se analiza en la Sección 2.3.2

Varias técnicas diferentes de detección de anomalías y análisis de los dataset para los sistemas IDS se han efectuado en el estado del arte. En [36] Ghazali utiliza cinco algoritmos de aprendizaje de máquina, y ambos dataset de la base de datos NSL-KDD (aprendizaje y evaluación) son mezclados y dividos en 10 subecciones; 9 de las subsecciones son utilizadas cómo dataset de entrenamiento y 1 es utilizada cómo dataset de evaluación. La mayor exactitud de la predicción del IDS fue conseguida con el clasificado PART, rama de los algoritmos de Árboles de Decision, con una exactitud del 96.7 %, sin embargo, teniendo en cuenta que el algoritmo PART fue entrenado y evaluado con vectores de ataques que eran previamente conocidos, su exactitud es la esperada. En [37] se propone un sistema IDS basado en anomalías con maquinas virtuales alojadas en un servidor en internet, teniendo una aproximación de un sistema IDS basado en la nube. La mejor exactitud en sus resultados fue conseguida con el algoritmo clasificador de aprendizaje de máquina Naive-Bayes, con una exactitud del 99.6 %, sus experimentos fueron

realizados con únicamente un 10 % de ambos dataset de NSL-KDD, sin embargo no se realiza ningún experimento con el dataset NSL-KDD completo, ni tampoco se menciona cual es la distribución de probabilidad de los ataques al hacer uso únicamente del 10 % de NSL-KDD.

Cómo se observa en los artículos [18, 36, 37], la mayoría de los trabajos en la actualidad usan el mismo dataset para el entrenamiento y evaluación de los algoritmos de aprendizaje de máquina mediante la división del dataset en dos partes, por lo tanto no hay vectores de ataques que no se encuentren en el dataset de evaluación, es decir, aquellos vectores de ataques de los cuales al algoritmo de aprendizaje de máquina no sean de su conocimiento previo al momento de evaluarse. *Por ésta causa* la mayoría de los trabajos en el estado del arte muestran resultados de incluso el 90 % aún sin un preprocesado de la base de datos de entrenamiento y evaluación, y de hasta el 99.9 % en caso de haberse realizado un tipo de preprocesado y pese a tener dimensiones reducidas. Éstos resultados *no* se consideran representativos debido a que los algoritmos de aprendizaje de máquina no son evaluados con ataques previamente desconocidos y por tanto no corresponden con el objetivo de los sistemas NIDS el cual es detectar ataques de los cuales no se tenga registro previo.

En éste trabajo de tesis se hace uso del dataset NSL-KDD-Training (obtenido mediante [9], y cuyo nombre de archivo es: *KDDTrain+*) para el entrenamiento de los clasificadores y además del dataset NSL-KDD-Testing (obtenido mediante [9], y cuyo nombre de archivo es: *KDDTest+*) para su evaluación, principalmente porque NSL-KDD-Testing contiene ataques adicionales los cuales no se encuentran en NSL-KDD-Training e imita nuevos tipos de ataques en la red, detección primordial en los sistemas NIDS basados en anomalías. De la misma manera más de un clasificador y algoritmo para reducción de dimensiones son utilizados para analizar el impacto en su exactitud y tiempos computacionales tanto de evaluación cómo de entrenamiento.

2.3.1. KDD99

Uno de los más grandes retos en los sistemas de detección de intrusos basados en red, es la cantidad masiva de datos recolectados del tráfico de la red, por ésta razón es necesario que antes de que cualquier tipo de información sea enviada a algún algoritmo de aprendizaje de máquina, el tráfico de la red en su estado binario después de algún volcado TCPDump (también llamado estado crudo), debe estar resumido en eventos de mayor nivel denominados registros de conexión, dónde cada registro de conexión es descrito mediante un conjunto de características (en inglés features), también llamadas dimensiones por su representación vectorial. Seleccionar características adecuadas es una actividad crucial, no trivial y requiere un extenso dominio del conocimiento de redes de datos.

En el año de 1999 el tráfico de red almacenado en la base de datos DARPA 98 [38] creada por los laboratorios Lincoln, fue resumido en conexiones de red con *41 características/dimensiones* por conexión. Después de éste preprocesado se formó lo que se considera el punto de referencia para la evaluación de los sistemas IDS, la base de datos KDD99, en el concurso internacional de descubrimiento de conocimiento y herramientas de minería de datos (en inglés Knowledge and Discovery Data Mining). Aún con los inconvenientes expuestos en [8], la base de datos KDD99 99 proporciona el único dataset original etiquetado, públicamente disponible para la comparación de los sistemas IDS.

Las bases de datos KDD99 de entrenamiento y evaluación usadas para la detección de intrusos están basados en la iniciativa DARPA del año 1998, la cual proporciona a los diseñadores de los sistemas de detección de intrusos un punto de referencia en el cual evaluar diferentes metodologías [38]. Para hacerlo, se realizó una simulación en una red militar ficticia consistiendo de tres máquinas objetivo corriendo diferentes sistemas operativos y servicios; adicionalmente tres máquinas fueron usadas después para suplantar diferentes direcciones IP con el objetivo de generar tráfico. Finalmente, hay un sniffer que registra todo el tráfico de red usando el formato TCPDump. El periodo total simulado es de siete semanas y cada uno de los registros de conexión es etiquetado dependiendo de su tipo en 5 clases principales. Las conexiones del tipo Normal son creadas al perfil normal del tráfico correspondiente a una red militar y los 41 tipos de ataques que ella contiene caen en una de las siguientes cuatro clases, Usuario a Administrador (U2R), Usuario remoto a usuario local (R2L), Negación de servicio (DoS) y Probe. Se describen de forma detallada a continuación.

- **Negación de Servicio (DoS).** El atacante envía de forma continua falsas solicitudes a un servidor, agotando los recursos de memoria, y por lo tanto el servidor comienza a rechazar solicitudes legitimas de los usuarios.
- **Usuario Remoto a Local (R2L).** El atacante envía paquetes a una máquina a través de la red sin que él posea una cuenta en dicha máquina, tratando de explotar alguna vulnerabilidad con el objetivo de ganar acceso a ella.
- **Usuario a Administrador (U2R).** El atacante tiene acceso local a través de una cuenta y trata de ganar privilegios de administrador.
- **Probe.** El atacante intenta conseguir información sobre la taxonomía de la red de computadoras que desea vulnerar con el propósito de eludir los controles de seguridad.

Los archivos originales en formato TCPDump de la competencia KDD, fueron preprocesados para la generación de la base de datos KDD99 y para

hacerlo los paquetes de información en los archivos TCPDump son resumidos en conexiones, dónde específicamente, "una conexión es una secuencia de paquetes TCP los cuales empiezan y terminan en algún tiempo bien definido, entre el cual la información fluye de una dirección IP origen a una dirección IP destino, bajo algún protocolo bien definido" [38]; cada registro de conexión consiste en aproximadamente 100 bytes. Éste proceso fue completado usando el software de código abierto Bro IDS [32], lo que resulta en 41 características/dimensiones por cada conexión. De las 41 caracteristicas se detallan cada una de ellas en la tabla 2.3 de la página 26, dónde el nombre se muestra en inglés para evitar algún posible problema de nomenclatura tal cual se muestra en la competencia original KDD [38], se agrega, además, una breve descripción y su tipo, continuo o nominal; sumado a ésto, las características de cada conexión están agrupadas en cuatro categorías siguientes:

1. **Características Básicas:** Son obtenidas de las cabeceras de los paquetes TCP sin inspeccionar el payload; las características básicas son las primeras seis mostradas en la tabla 2.3.

2. **Características de Contenido:** Éstas características incluyen valores tales cómo el número de intentos fallidos de logueo.

3. **Características de tráfico basadas en el tiempo:** Éstas características están diseñadas para capturar propiedades que maduran sobre una ventana temporal de 2 segundos. Un ejemplo de dicha característica es el número de conexiones del mismo host sobre el intervalo de 2 segundos.

4. **Características de tráfico basadas en Host:** Utiliza una ventana histórica de tiempo estimada sobre el número de conexiones (en éste caso 100) en vez del tiempo. Las características basadas en host son por consiguiente diseñadas para evaluar ataques los cuales abarcan intervalos de tiempo mayores a 2 segundos.

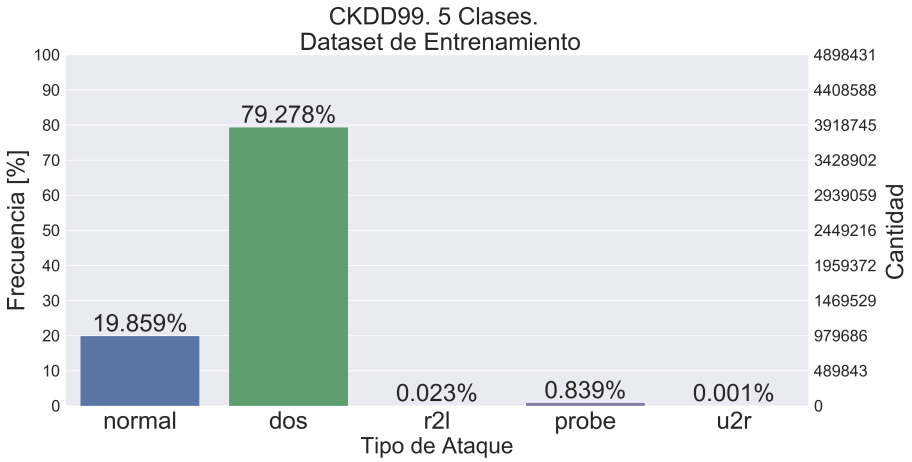
La base de datos KDD99 se encuentra para descarga pública en [7]; se muestra dividida en tres bases de datos principales, cuyos nombres se mencionan en la tabla 2.1 de manera idéntica a los encontardos en su página web. Dos de las bases de datos KDD99 están diseñadas para el entrenamiento de los sistemas IDS, "KDD Cup Data" la cual cuenta con aproximadamente cuatro millones de vectores y, "KDD Cup Data 10 Percent" con aproximadamente 400000 vectores. Al mismo tiempo cuenta con una base de datos diseñada para la evaluación de la exactitud y rendimiento de los sistemas IDS, "Corrected KDD" con aproximadamente 300000 vectores. La tabla 2.1 muestra la cantidad de vectores etiquetados en alguna de las cinco clases principales, DoS, U2R, R2L, Probe y Normal correspondientes a las tres

bases de datos de KDD99; por cuestiones de practicidad en el contenido restante de la tesis, a la base de datos "KDD Cup Data" se le hará mención con el nombre de "CKDD99" dónde la C al inicio representa la palabra "Completa", a "KDD Cup Data 10 Percent" se le hará mención con el nombre de "10KDD99" y a "Corrected KDD" se le denominará "EKDD99" dónde la letra "E" al inicio representa la palabra "Evaluación".

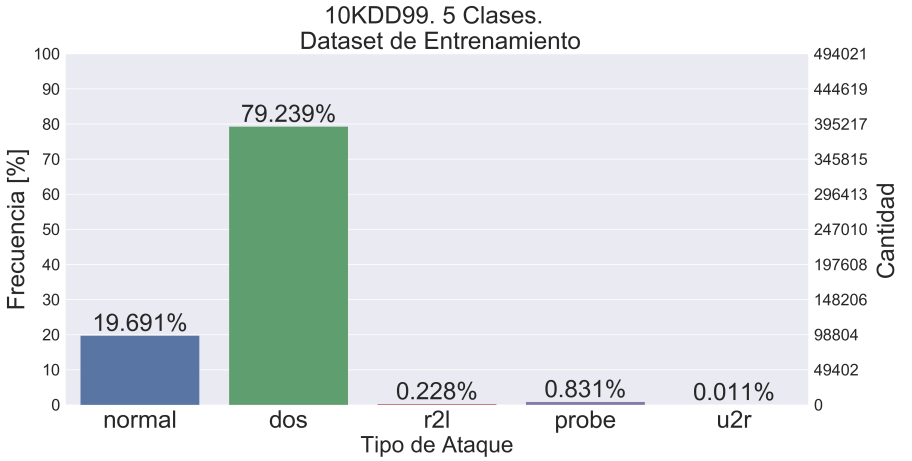
Tabla 2.1: Bases de datos de KDD99

Dataset	Dos	Probe	U2R	R2L	Normal	Total
KDD Cup Data	3883370	41102	52	1126	972781	4898431
KDD Cup Data 10 Percent	391458	4107	52	1126	97278	494021
Corrected KDD	229853	4166	228	16189	60593	311029

Para expandir y ejemplificar la tabla 2.1, la Figura 2.4 muestra la distribución de las cinco clases principales en las base de datos de entrenamiento KDD99, además de su porcentaje en relación a la totalidad del dataset; la Figura 2.4a muestra la distribución de cada clase para la base de datos completa (CKDD99) y la Figura 2.4b muestra la distribución de cada clase para el 10 % de la base de datos KDD99 (10KDD99). A simple vista es posible observar lo mencionado por J. McHugh [8] en relación a la cantidad enorme de instancias de la clase normal y al tipo de ataque específico DoS, al mismo tiempo con menos del 0.01 % del total del dataset para las clases R2L y U2R, lo que provoca resultados sesgados hacia aquellos clasificadores en los algoritmos de ML los cuales tengan una mayor habilidad para encontrar patrones que se repiten múltiples veces. La Figura 2.5 muestra la distribución de las cinco clases para la base de datos de evaluación de los sistemas IDS, EKDD99, en ella se mantiene la cantidad superior de instancias para la clase DoS y Normal, además de un incremento para las clases R2L y Probe, ésto debido a los tipos de ataques nuevos que no se encuentran en las bases de datos de entrenamiento, éstos ataques junto con su cantidad de instancias se muestran en la tabla 2.2.



(a) Distribución de las cinco clases principales en la base de datos CKDD99



(b) Distribución de las cinco clases principales en la base de datos 10KDD99

Figura 2.4: Distribución de las cinco clases principales en las bases de entrenamiento IDS, CKDD99 y 10KDD99

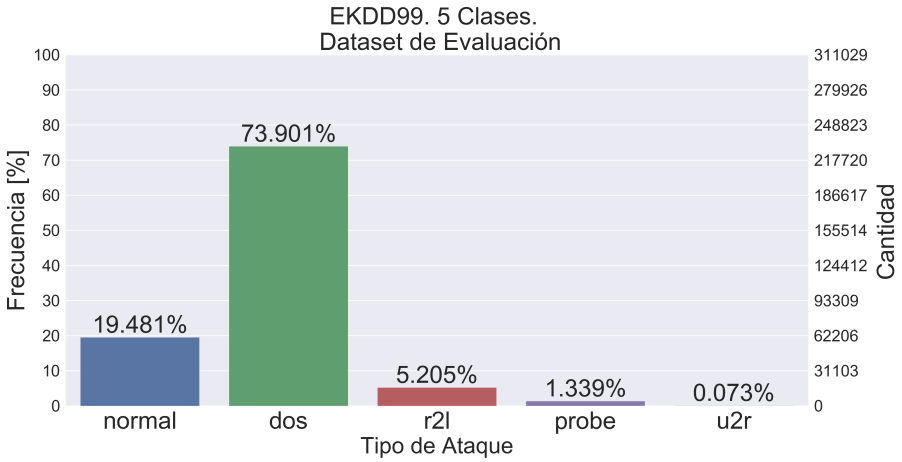


Figura 2.5: Distribución de las cinco clases principales en la base de datos EKDD99

Con el objeto de comparar la exactitud de los algoritmos de aprendizaje de máquina aplicados en los sistemas IDS, es imprescindible que la base de datos que sea utilizada para evaluación posea vectores considerados y por lo tanto etiquetados cómo ataques/anomalías, los cuales no se encuentren en la base de datos de entrenamiento de los algoritmos de aprendizaje de máquina, de forma que se cumpla la razón de los sistemas IDS, al ser enfrentados y por lo tanto detectar intrusiones de las cuales no se tiene un conocimiento previo.

Las bases de datos de entrenamiento utilizadas para los sistemas IDS, CKDD99 y 10KDD99, cuentan con un total de 22 tipos de ataques diferentes, etiquetados en alguna de las cuatro clases principales: DoS, U2R, R2L y Probe. La base de datos de evaluación de los sistemas IDS, EKDD99, cuenta con 17 tipos de ataques nuevos los cuales no se encuentran en CKDD99 ni 10KDD99, e igualmente etiquetados en alguna de las cuatro clases principales. La tabla 2.2 muestra el total de ataques existentes en los dos dataset de entrenamiento y en el dataset de evaluación; muestra cada uno de los 39 ataques totales, junto con su nombre, la clase principal a la que pertenecen y el total de instancias en las tres bases de datos. La descripción completa de cada uno de los ataques se puede encontrar la tesis de Robert Kendall en [39].

Desde la Competencia Internacional de Descubrimiento de Conocimiento y Herramientas de Minado de Datos (KDD98) y en algunos de los trabajo del estado del arte contemporáneos, la base de datos KDD99 es utilizada para el entrenamiento y evaluación de los sistemas IDS basados en anomalías, pese a las deficiencias mostradas y expuestas en [8] y [9]. Los trabajos actuales que aún hace uso de KDD99 no utilizan a CKDD99 para el entrenamiento de los

IDS, sino su versión reducida 10KDD99, debido al alto costo computacional que demanda realizar un preprocesado y entrenamiento de los clasificadores con un archivo .csv de 4 millones de instancias.

En el año 2009 Mahbod y su equipo [9] proponen una base de datos de entrenamiento y evaluación de los sistemas IDS, NSLKDD, la cual está basada y soluciona algunos de los problemas en KDD99, dónde si bien las instancias de las clases DoS y Normal se mantienen cómo la mayoría del dataset, las demás clases aumentan su cantidad de instancias, mejorando su distribución de probabilidad. NSLKDD es la base de datos para el entreniento y evaluación de los sistemas IDS con mayor indice de uso en la literatura actual, la única disponible públicamente y aceptada por los investigadores, de lo antedicho se desprende que NSLKDD es utilizada para el entrenamiento y evaluación de los sistemas IDS en éste trabajo de tesis. NSLKDD se explica desde su generación y se detalla en el capítulo 2.3.2.

Tabla 2.2: Tipos de ataques y su distribución en KDD99

No.	Ataque	Clase	CKDD99	10KDD99	EKDD99
1.	back	DOS	2203	2203	1098
2.	land	DOS	21	21	9
3.	neptune	DOS	1072017	107201	58001
4.	pod	DOS	264	264	87
5.	teardrop	DOS	979	979	12
6.	smurf	DOS	2807886	280790	164091
7.	apache2	DOS	0	0	794
8.	mailbomb	DOS	0	0	5000
9.	processtable	DOS	0	0	759
10.	udpstorm	DOS	0	0	2
11.	buffer overflow	U2R	30	30	22
12.	loadmodule	U2R	9	9	2
13.	perl	U2R	3	3	2
14.	rootkit	U2R	10	10	13
15.	httptunnel	U2R	0	0	158
16.	ps	U2R	0	0	16
17.	sqlattack	U2R	0	0	2
18.	xterm	U2R	0	0	13
19.	guess passwd	R2L	53	53	4367
20.	ftp write	R2L	8	8	3
21.	imap	R2L	12	12	1
22.	multihop	R2L	7	7	18
23.	phf	R2L	4	4	2
24.	warezmaster	R2L	20	20	1602
25.	warezclient	R2L	1020	1020	0
26.	spy	R2L	2	2	0
27.	named	R2L	0	0	17
28.	sendmail	R2L	0	0	17
29.	snmpgetattack	R2L	0	0	7741
30.	snmpguess	R2L	0	0	2406
31.	worm	R2L	0	0	2
32.	xlock	R2L	0	0	9
33.	xsnoop	R2L	0	0	4
34.	satan	Probe	15892	1589	1633
35.	ipsweep	Probe	12481	1247	306
36.	nmap	Probe	2316	231	84
37.	portsweep	Probe	10413	1040	354
38.	mscan	Probe	0	0	1053
39.	saint	Probe	0	0	736
40.	Total:		3925650	396743	250436

Tabla 2.3: Descripción de las 41 características en el dataset KDD99

No.	Nombre	Descripción	Tipo (C/N)
1.	Duration	Longitud de la conexión	Continuo
2.	Protocol Type	Tipo de protocolo (ej. TCP, UDP)	Nominal
3.	Service	Servicio de destino (ej. Telnet, FTP)	Nominal
4.	Flag	Estado de la bandera de la conexión	Nominal
5.	Src Bytes	Bytes enviados de la fuente al destino	Continuo
6.	Dst Bytes	Bytes enviados del destino a la fuente	Continuo
7.	Land	1 Si la conexión es desde/a el mismo host/puerto; 0 en otro caso	Nominal
8.	Wrong Fragment	Número de Fragmentos erroneos	Continuo
9.	Urgent	Número de paquetes urgentes	Continuo
10.	Hot	Número de indicadores del tipo HOT	Continuo
11.	Num Failed Logins	Número de intentos fallados de logueo	Continuo
12.	Logged In	1 si el logueo se consiguió 0 en otro caso	Nominal
13.	Num Compromised	Número de condiciones “comprometidas”	Continuo
14.	Root Shell	1 si se consigue una consola de administardor 0 en otro caso	Nominal
15.	Su Attempted	1 si se intentó el comando “su root” 0 en otro caso	Nominal
16.	Num Root	Número de accesos “Root”	Continuo
17.	Num File Creations	Número de operaciones de creaciones de archivos	Continuo
18.	Num Shells	Número de consolas shell	Continuo
19.	Num Access Files	Número de operaciones en el acceso de archivos de control	Continuo
20.	Num Outbound Cmds	Número de comandos salientes en una sesión FTP	Continuo
21.	Is Hot Login	1 Si el logueo pertenece a “Hot” 0 en otro caso	Nominal
22.	Is Guest Login	1 si es el logueo de un “invitado”; 0 en otro caso	Nominal
23.	Count	Número de conexiones al mismo host cómo la conexión actual en los últimos 2 seg	Continuo
24.	Srv Count	Número de conexiones al mismo servicio cómo la conexión actual en los últimos 2 seg	Continuo
25.	Serror Rate	% de conexiones que tienen errores “SYN” Conexiones al mismo Host	Continuo
26.	Srv Serror Rate	% de conexiones que tienen errores “SYN” Conexiones al mismo Servicio	Continuo
27.	Rerror Rate	% de conexiones que tienen errores “REJ” Conexiones al mismo Host	Continuo
28.	Srv Rerror Rate	% de conexiones que tienen errores “REJ” Conexiones al mismo Servicio	Continuo
29.	Same Srv Rate	% de conexiones al mismo servicio (Conexiones a mismo servicio)	Continuo
30.	Diff Srv Rate	% de conexiones a diferentes servicios	Continuo
31.	Srv Diff Host Rate	% de conexiones a diferentes Host (Conexiones al mismo servicio)	Continuo
32.	Dst Host Count	Conteo de conexiones teniendo el mismo host de destino	Continuo
33.	Cst Host Srv Count	Conteo de conexiones que tienen el mismo host de destino y usan el mismo servicio	Continuo
34.	Dst Host Same Srv Rate	% de xonexiones teniendo el mismo host de destino y usando el mismo servicio	Continuo
35.	Dst Host Diff Srv Rate	% de diferentes servicios en el host actual	Continuo
36.	Dst Host Same Src Port Rate	% de conexiones al host actual las cuales tienen el mismo puerto fuente	Continuo
37.	Dst Host Srv Diff Host Rate	% de conexiones al mismo servicio viniendo de diferentes host	Continuo
38.	Dst Hot serror Rate	% de conexiones al Host actual que tienen un error S0	Continuo
39.	Dst Host Srv Serror Rate	% de conexiones al Host actual y servicio especificado que tienen un error S0	Continuo
40.	Dst Host Rerror Rate	% de conexiones al Host actual que tienen un error RST	Continuo
41.	Dst Host Srv Rerror Rate	% de conexiones al host actual y servicio especificado que tienen un error RST	Continuo
42.	Etiqueta	Etiqueta correspondiente a las clases: Normal o algún tipo de ataque	Nominal

2.3.2. NSL-KDD99

NSL-KDD99 [40] es una base de datos en formato ".csv" y "arff" utilizada para el entrenamiento y evaluación de los sistemas NIDS basados en anomalías. Cuenta con una base de datos de entrenamiento, "KDDTrain+", con 125973 registros de conexión, y una base de datos de evaluación, "KDDTest+", con 22543 registros. NSLKDD es propuesta en el año 2009 y está basada en la base de datos KDD99, compartiendo las 41 caracteristicas/dimensiones pertenecientes al tráfico de red de la tabla 2.3 y los 39 tipos de ataques descritos en la tabla 2.2.

NSL-KDD99 es propuesta por Tavallaee y su equipo [9], diseñada para solventar algunos de los problemas subyacentes de la base de datos KDD99 los cuales se discuten a continuación y a detalle por McHugh [8] y Mahoney [41].

2.3.2.1. Problemáticas de la base de datos KDD99

En [9], los autores realizan un análisis estadístico de las tres bases de datos pertenecientes a KDD99, lo cual los lleva a encontrar dos defectos principales, el primero es el gran número de instancias redundantes, el segundo es el análisis del nivel de dificultad de las instancias en KDD99. Su estudio muestra que hay, además, deficiencias más importantes en los propios dataset del mismo conjunto de datos de entrenamiento y evaluación; siendo uno de éstos problemas el etiquetado de forma errónea de los registros de conexión, ya que existen 9 registros de conexión etiquetados cómo diferentes ataques y hay otros 9 registros con caracteristicas idénticas etiquetados cómo conexión normal. Otro problema son las características cuyo valor permanece inalterado en todo el dataset, por ejemplo, la dimensión número 20, perteneciente al número de comandos salientes en una sesión FTP, tiene el valor de 0 para todos los registros de conexión, tanto en la base de datos de entrenamiento cómo la base de datos de evaluación, éste problema se describe más adelante en la sección 2.4.1

El primer defecto principal en KDD99 es el número de instancias redundantes (duplicadas). En [9] los autores proponen una solución para reducir el problema de la inmensa cantidad de instancias repetidas, por ésta razón eliminan todos los registros repetidos pertenecientes a las dos bases de entrenamiento, CKDD99 y 10KDD99, así cómo a la base de datos de evaluación EKDD99, manteniendo unicamente una copia de cada registro de conexión.

La Figura 2.6 muestra un ejemplo de los registros de conexión redundantes en la base de datos CKDD99, dónde el registro de conexión etiquetado cómo el tipo de ataque "back" perteneciente a la clase "DoS" se repite al menos en tres lineas, de la misma manera para el registro de conexión del tipo normal. Éstos patrones se pueden observar con múltiples ataques en múltiples instancias y no únicamente los mostrados en la Figura 2.6, los cuales

son demostrativos. En el apéndice A.2 se muestra el código realizado para la búsqueda de éstos registros de conexión.

%Ejemplo de registros de conexión redundantes con la clase del tipo: Dos
Registro de Conexión No: 499534
**0,tcp,http,SF5,4540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,0,1,2,0.0,0.0,0.0,0.5,1.0,0.0,1.0,255,255,1.0,0.0,0.0,0.0,0.01,0.01,0.04,0.04,back,dos,1,1,1
Registro de Conexión No: 499540
**0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,0,1,2,0.0,0.0,0.0,0.0,0.5,1.0,0.0,1.0,255,255,1.0,0.0,0.0,0.0,0.01,0.01,0.04,0.04,back,dos,1,1,1
Registro de Conexión No: 499550
**0,tcp,http,SF,54540,8314,0,0,0,2,0,1,1,0,0,0,0,0,0,0,0,0,0,1,2,0.0,0.0,0.0,0.0,0.5,1.0,0.0,1.0,255,255,1.0,0.0,0.0,0.0,0.01,0.01,0.04,0.04,back,dos,1,1,1

%Ejemplo de registros de conexión redundantes con la clase del tipo: Normal
Registro de Conexión No: 3447955
**20226,udp,other,SF146,105,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255,1,0.0,0.85,1.0,0.0,0.0,0.0,0.0,0.0,0.0,normal,normal,16,2,0
Registro de Conexión No: 3447967
**20226,udp,other,SF146,105,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255,1,0.0,0.85,1.0,0.0,0.0,0.0,0.0,0.0,0.0,normal,normal,16,2,0
Registro de Conexión No: 3447982
**20226,udp,other,SF146,105,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255,1,0.0,0.85,1.0,0.0,0.0,0.0,0.0,0.0,0.0,normal,normal,16,2,0

Figura 2.6: Instancias Redundantes en CKDD99

Otro problema de KDD99 pertenece a la base de datos de evaluación EKDD99, la cual contiene en total 311029 registros de conexión y dos de ellos contienen el parámetro "icmp" en la dimensión número 3, correspondiente al tipo de servicio; éstos registros fueron removidos en [9].

Paralelamente se encontraron 70 registros de conexión etiquetados cómo diferentes tipos de ataques y otras conexiones idénticas teniendo los mismos valores en sus dimensiones etiquetadas cómo conexión normal. Un ejemplo de los registros de conexión que comparten los mismos valores en cada una de sus características y son etiquetados de forma diferente se muestra en la Figura 2.7 para el caso de la base de datos EKDD99; ésto hace referencia a lo mencionado por McHuge [8], sobre la falta de una definición concreta y unequivoca para los registros de conexión que son considerados cómo ataques. Todos éstos registros son removidos en [9]

%Registros de conexión idénticos etiquetados cómo: Normal y Satan
Linea No: 8372
0,tcp,other,REJ0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,1,0.06,0.0,0.94,1.0,0.0,0.96,0.0,255,1,0
.0,0.91,0.0,0.0,0.03,0.0,0.97,1.0,normal,normal
Linea No: 8415
0,tcp,other,REJ0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,511,1,0.05,0.0,0.94,1.0,0.0,0.96,0.0,255,1,0
.0,0.91,0.0,0.0,0.03,0.0,0.97,1.0,satan,probe
%Registros de conexión idénticos etiquetados cómo: Normal y Pod
Linea No: 40897
0,udp,private,SF105,147,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255,25
0,0.98,0.01,0.0,0.0,0.0,0.0,0.0,0.0,normal,normal
Linea No: 40903
0,udp,private,SF105,147,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0.0,0.0,0.0,0.0,1.0,0.0,0.0,255,25
0,0.98,0.01,0.0,0.0,0.0,0.0,0.0,0.0,pod,dos

Figura 2.7: Registros de conexión idénticos con múltiples etiquetas en la base de datos EKDD99

2.3.3. Taxonomía de NSL-KDD99

NSL-KDD99 se encuentra disponible para los investigadores a través de [40], sin embargo, el acceso para descarga debe ser pedido personalmente a la dirección de correo electrónico a.habibi.l@unb.ca. NSL-KDD99 cuenta con siete tipos de archivos diferentes, divididos en formato "arff" y "csv". En éste trabajo de tesis se hace uso de los tipos de archivo "csv", específicamente de la base de datos de entrenamiento "KDDTrain+" y de la base de datos de evaluación "KDDTest+". Por cuestiones de practicidad en éste trabajo de tesis a la base de datos "KDDTrain+" se le denomina "NSLKDD" y a la base de datos de evaluación "KDDTest+" se le hace referencia con el nombre de "ENSLKDD". La tabla 2.4 muestra de forma general la distribución de las cinco clases principales en los registros de conexión en la base de datos de entrenamiento NSLKDD y de evaluación ENSLKDD.

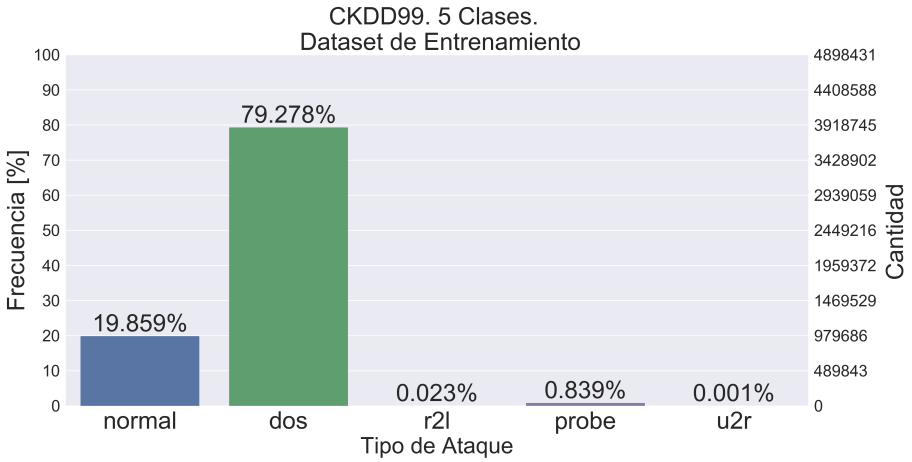
Tabla 2.4: Bases de datos de NSL-KDD99

Dataset	Dos	Probe	U2R	R2L	Normal	Total
NSLKDD	45927	11656	52	995	67343	125973
ENSLKDD	7458	2421	200	2754	9710	22543

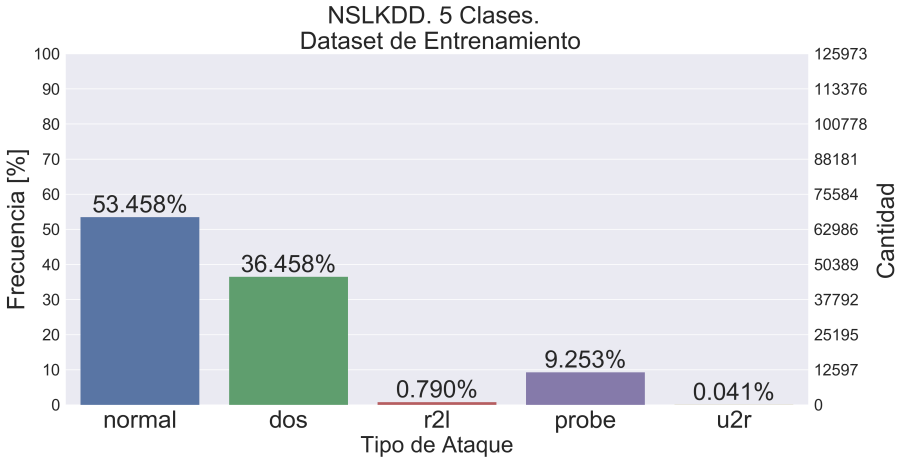
De la tabla 2.4 es posible observar que las distribuciones de probabilidad para las cinco clases en NSL-KDD99 cambian de manera abrupta y al mismo tiempo el total de instancias en NSLKDD y ENSLKDD se ve reducido en contraste con la base de datos KDD99 mostrada en la tabla 2.1. El número de registros de conexión en la base de datos de entrenamiento NSLKDD es de 125973 y en la base de datos de evaluación ENSLKDD es de 22543 registros, las cuales son cantidades por más razonables a comparación de los 4 millones

de registros de CKDD99 e incluso los 400000 registros en 10KDD99. Ésta ventaja de NSL-KDD99 hace posible realizar experimentos con la base de datos completa, sin la necesidad de seleccionar porciones aleatorias de las bases de datos de entrenamiento y evaluación cómo en diversos trabajos del estado del arte. En consecuencia los resultados obtenidos haciendo uso de NSL-KDD99 por investigadores pueden ser consistentes y comparables, al eliminar la aleatoriedad de la selección de registros en una pequeña parte del dataset. De la misma manera que en CKDD99 y EKDD99 existen registros de conexión etiquetados cómo ataques los cuales no se encuentran en ambos dataset, y éstos se detallan más adelante.

Cómo se ha comentado en [9] los autores solucionan los problemas de los registros de conexión contenidos en KDD99 mostrados en la sección 2.3.2.1, y al mismo tiempo, solucionan la problemática del nivel de dificultad. Ésta problemática reside en el hecho de que la típica aproximación para la realización de los sistemas NIDS basados en anomalías es haciendo uso de la base de datos KDD99 a través de la cual se emplea un algoritmo de aprendizaje de máquina para aprender el comportamiento general de la red a través de la base de datos de entrenamiento de manera que sea capaz de diferenciar entre actividades normales y maliciosas. Para ésto en [9] se hizo un estudio de la exactitud en la clasificación de diferentes algoritmos de aprendizaje de máquina entrenados usando KDD99 y se encontró que pese a no realizar ningún preprocesado a la base de datos ni modificar parámetros propios del algoritmo de aprendizaje de máquina, el resultado de algunos algoritmos era de una exactitud de hasta el 90 %, parámetro muy alto y no conseguible en un escenario real. Para solucionarlo Tavallaee mejora la distribución de probabilidad de las cinco clases, de forma que no estén completamente centralizadas en los registros de conexión del tipo DoS y Normal en contraste con KDD99. La Figura 2.8 muestra la comparativa en la distribución de probabilidad de las cinco clases principales para las bases de datos de entrenamiento, NSLKDD y CKDD99, dentro de la cual la Figura 2.8a muestra su distribución para la base de datos CKDD99 y la Figura 2.8b muestra su distribución para la base de datos NSLKDD. Es posible observar una mejora en la distribución de proabilidad en NSLKDD en contraste con CKDD99 disminuyendo la clase del tipo de ataque DoS así cómo un aumento significativo para las clases R2L y Probe. U2R se mantiene con un porcentaje bajo debido a que la base de datos completa CKDD99 tiene un total de 52 registros de conexión pertenecientes a ese ataque, los cuales se muestran en la tabla 2.1; éstos 52 registros de conexión de la clase U2R se mantienen en NSLKDD y es posible observarlos a través de la tabla 2.4.



(a) Distribución de probabilidad de las cinco clases principales en CKDD99

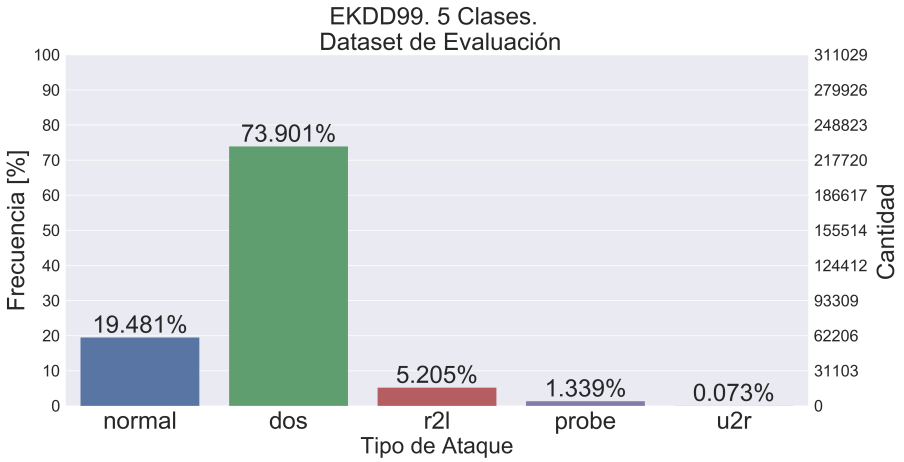


(b) Distribución de probabilidad de las cinco clases principales en NSLKDD

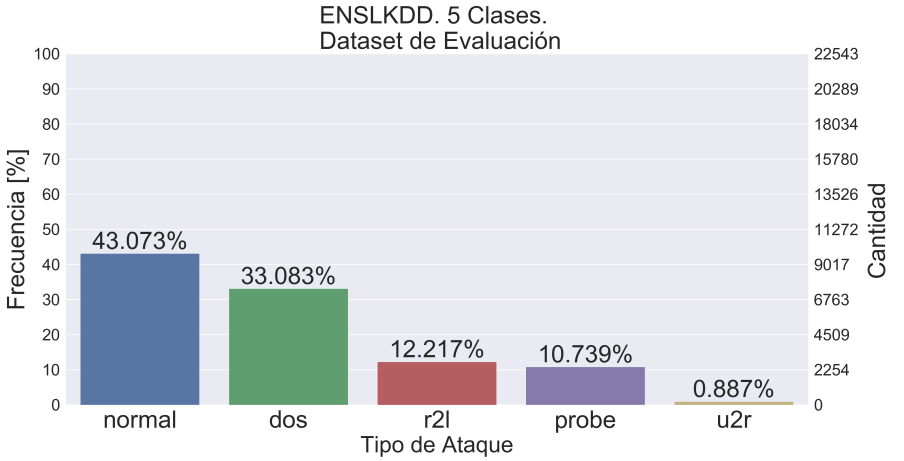
Figura 2.8: Comparativa de la distribución de probabilidad de las cinco clases principales en CKDD99 y NSLKDD

.

La Figura 2.9 muestra la comparativa en la distribución de probabilidad de las cinco clases principales para las bases de datos de evaluación, EKDD99 y ENSLKDD, la Figura 2.9a muestra su distribución para la base de datos EKDD99 y la Figura 2.9b muestra su distribución para la base de datos ENSLKDD. És posible observar una mejor distribución de probabilidad para las 5 clases, no sólo siendo enfocado el dataset para la clase del tipo DoS, lo que soluciona el problema de las altas tasas de exactitud de algunos clasificadores sin ningún preprocesado.



(a) Distribución de probabilidad de las cinco clases principales en EKDD99



(b) Distribución de probabilidad de las cinco clases principales en ENSLKDD

Figura 2.9: Comparativa de la distribución de probabilidad de las cinco clases principales en EKDD99 y ENSLKDD

.

De igual manera que al estudio realizado a KDD99 y del cual se obtiene la tabla 2.2 la cual muestra el total de ataques contenidos en CKDD99 y EKDD99, la tabla 2.5 muestra el total de ataques contenidos en NSLKDD y ENSLKDD. Paralelamente que en EKDD99, ENSLKDD contiene ataques nuevos los cuales no se encuentran en la base de datos ENSLKDD, con el fin de evaluar el sistema NIDS basado en anomalías frente a ataques desconocidos.

Tabla 2.5: Tipos de ataques y su distribución en NSL-KDD99

No.	Ataque	Clase	NSLKDD	ENSLKDD
1.	back	DOS	956	359
2.	land	DOS	18	7
3.	neptune	DOS	41214	4657
4.	pod	DOS	201	41
5.	teardrop	DOS	892	12
6.	smurf	DOS	2646	665
7.	apache2	DOS	0	737
8.	mailbomb	DOS	0	293
9.	processtable	DOS	0	685
10.	udpstorm	DOS	0	2
11.	buffer overflow	U2R	30	20
12.	loadmodule	U2R	9	2
13.	perl	U2R	3	2
14.	rootkit	U2R	10	13
15.	httptunnel	U2R	0	133
16.	ps	U2R	0	15
17.	sqlattack	U2R	0	2
18.	xterm	U2R	0	13
19.	guess passwd	R2L	53	1231
20.	ftp write	R2L	8	3
21.	imap	R2L	11	1
22.	multihop	R2L	7	18
23.	phf	R2L	4	2
24.	warezmaster	R2L	20	944
25.	warezclient	R2L	890	0
26.	spy	R2L	2	0
27.	named	R2L	0	17
28.	sendmail	R2L	0	14
29.	snmpgetattack	R2L	0	178
30.	snmpguess	R2L	0	331
31.	worm	R2L	0	2
32.	xlock	R2L	0	9
33.	xsnoop	R2L	0	4
34.	satan	Probe	3633	735
35.	ipsweep	Probe	3599	141
36.	nmap	Probe	1493	73
37.	portsweep	Probe	2931	157
38.	mscan	Probe	0	996
39.	saint	Probe	0	319
40.	Total:		58630	12833

2.4. Preprocesado

Antes de que las bases de datos sean utilizadas para el entrenamiento de los los algoritmos de aprendizaje de máquina en los sistemas NIDS es necesario que a éstas sean aplicadas métodos de preprocesado, con el fin de que su representación sea más adecuada. Los métodos de preprocesado inician con técnicas para una correcta estructura de la base de datos, puesto que a menudo es observado que el 80 % del análisis de datos es gastado en el proceso del limpiado y preparación de la información [42]. Hadley en [43] menciona el concepto de "Tidy Data" (Datos Ordenados, en español) dónde establece tres puntos importantes en la estructura de cualquier base de datos:

1. Cada variable/característica debe formar una columna.
2. Cada observación/instancia debe formar una fila.
3. Cada tipo de unidad observacional debe formar una tabla.

El punto número 3 hace referencia al hecho de que no es recomendable mantener dos tipos de elementos de estudio en la misma tabla, sino deben de permanecer en tablas diferentes. La Figura 2.10 muestra los cinco primeros registros de conexión en la base de datos NSL-KDD99, dentro de ella, la Figura 2.10a representa los primeros cinco registros para la base de datos NSLKDD y paralelamente 2.10b para la base de datos ENSLKDD. Ambas bases de datos cumplen con los criterios de Tidy Data.

	Duration	ProtocolType	Service	Flag	...	DstHostRerrorRate	DstHostSrvRerrorRate	TypeOfAttack	Group
0	0	tcp	ftp_data	SF	...	0.05	0.00	normal	normal
1	0	udp	other	SF	...	0.00	0.00	normal	normal
2	0	tcp	private	S0	...	0.00	0.00	neptune	dos
3	0	tcp	http	SF	...	0.00	0.01	normal	normal
4	0	tcp	http	SF	...	0.00	0.00	normal	normal

(a) Primeros cinco registros de conexión en NSLKDD

	Duration	ProtocolType	Service	Flag	...	DstHostRerrorRate	DstHostSrvRerrorRate	TypeOfAttack	Group
0	0	tcp	private	REJ	...	1.00	1.00	neptune	dos
1	0	tcp	private	REJ	...	1.00	1.00	neptune	dos
2	2	tcp	ftp_data	SF	...	0.00	0.00	normal	normal
3	0	icmp	eco_i	SF	...	0.00	0.00	saint	probe
4	1	tcp	telnet	RSTO	...	0.83	0.71	mscan	probe

(b) Primeros cinco registros de conexión en ENSLKDD

Figura 2.10: Tidy data en NSL-KDD99

.

2.4.1. Análisis de información y entropía

Seguido al análisis de la estructura de ENSLKDD y NSLKDD, se realiza un estudio del promedio, desviación estándar, información y entropía en cada una de las dimensiones no Nominales de la tabla 2.3, es decir, sus 41 dimensiones originales excepto 2,3,4,7,12,14,15,21 y 22. La formula de la desviación estándar se describe en la ecuación 2.1, dónde N representa el total de registros de conexión, y x_i un registro de conexión específico. La información se describe en la ecuación 2.2 y la entropía en 2.3, dónde X representa al conjunto de elementos pertenecientes a una dimensión específica, $P(x_i)$, la probabilidad de que suceda x_i valor en dicha dimensión, y $I(x_i)$ su información.
Éste estudio se realiza con el fin de determinar aquellas dimensiones que resultan redundantes las cuales no aportan ningún valor al momento de ser utilizadas por los algoritmos de ML.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \left(\frac{1}{N}\sum_{i=i}^{N} x_i\right)^2\right)} \tag{2.1}$$

$$I(X) = -\log_2 P(X) \tag{2.2}$$

$$H(X) = \sum_{i=1}^{N} P(x_i) I\left(x_i\right) = -\sum_{i=1}^{N} P\left(x_i\right) \log_2 P\left(x_i\right) \tag{2.3}$$

La tabla 2.6 resultado de hacer el análisis por desviación estándar y entropía en ENSLKDD y NSLKDD, arroja resultados importantes. Cómo se hizo mención en la sección 2.3.2.1, la dimensión número número 20 correspondiente al número de comandos salientes en una sesión FTP tiene una varianza de 0, es decir el valor numérico que ella contiene (0) permanece inalterado en ENSLKDD y NSLKDD, de manera que no ayuda en el análisis del clasificador, por lo tanto ésta dimensión se desprecia para el análisis de las siguientes secciones de éste trabajo de tesis; el resultado final son dos bases de datos de 40 dimensiones cada una.

Tabla 2.6: Análisis estadístico de NSL-KDD99

No.	Nombre	Promedio	Desviación Std	Información	Entropia
1.	Duration	218.868	1407.207	8557.617	1.560
5.	Src Bytes	10395.911	472796.912	14849.668	6.277
6.	Dst Bytes	2056.110	21219.763	50219.489	6.529
8.	Wrong Fragment	0.008	0.142	17.653	0.045
9.	Urgent	0.0007	0.036	39.059	0.006
10.	Hot	0.105	0.928	179.049	0.308
11.	Num Failed Logins	0.021	0.150	47.401	0.1502
13.	Num Compromised	0.119	7.269	296.178	0.136
16.	Num Root	0.114	8.041	261.550	0.029
17.	Num File Creations	0.008	0.676	102.521	0.023
18.	Num Shells	0.001	0.048	37.890	0.010
19.	Num Access Files	0.003	0.067	50.195	0.032
20.	Num Outbound Cmds	0.0	0.0	0.0	0.0
23.	Count	79.028	128.542	5441.791	6.019
24.	Srv Count	31.125	89.064	5604.800	4.935
25.	Serror Rate	0.102	0.295	1045.899	1.027
26.	Srv Serror Rate	0.103	0.298	990.036	0.837
27.	Rerror Rate	0.238	0.416	1048.979	1.398
28.	Srv Rerror Rate	0.235	0.416	1104.991	1.207
29.	Same Srv Rate	0.740	0.412	817.196	2.179
30.	Diff Srv Rate	0.094	0.259	1143.835	1.820
31.	Srv Diff Host Rate	0.098	0.253	879.617	1.812
32.	Dst Host Count	193.866	94.036	2518.196	3.575
33.	Cst Host Srv Count	140.756	111.782	2438.463	5.386
34.	Dst Host Same Srv Rate	0.608	0.435	836.302	4.123
35.	Dst Host Diff Srv Rate	0.090	0.220	969.766	3.488
36.	Dst Host Same Src Port Rate	0.132	0.306	1009.412	2.737
37.	Dst Host Srv Diff Host Rate	0.019	0.085	611.928	1.792
38.	Dst Hot serror Rate	0.097	0.273	1036.885	1.622
39.	Dst Host Srv Serror Rate	0.099	0.281	1093.617	1.250
40.	Dst Host Rerror Rate	0.233	0.387	902.155	2.965
41.	Dst Host Srv Rerror Rate	0.226	0.400	1012.609	1.971

2.4.2. Escalamiento

El escalamiento (Scaling, en inglés) también llamado normalización, es uno de los métodos de preprocesamiento para los algoritmos de aprendizaje de máquina. Su objetivo es escalar la información de manera que caiga dentro de un pequeño rango específico, generalmente entre $[-1, 1]$ para el caso de la normalización *Z-Score*, y entre $[0, 1]$ para el caso de la normalización *Min-Max*. Realizar un tipo de normalización antes de aplicar cualquier tipo de algoritmo de aprendizaje de máquina es efectivo en reducir el tiempo de aprendizaje de éstos algoritmos, así cómo su importancia, ésto debido al diferente rango de valores en cada una de las dimensiones en NSL-KDD99. De entre los algoritmos dónde la normalización tiene un efecto importante se encuentra, KNN, al tener que medir la distancia euclidiana y SVMs al tener que calcular los coeficientes para el hiperplano, todos ellos utilizados en éste trabajo de tesis y por lo antedicho es realizado el proceso de normalización

en NSL-KDD99. Los únicos algoritmos de ML que no se ven afectados por métodos de escalamiento, son aquellos derivados de los árboles de decisión.

Existen dos métodos de normalización mayormente utilizados, normalización *Z-Score* y *Min-Max* [44]

2.4.2.1. Min-Max

La normalización Min-Max llamada a veces simplemente normalización, lo cual causa ambigüedades ya que difiere de aquella comunmente utilizada dónde se encuentra el valor con la mayor magnitud de un vector, y todos los demás elementos que el contienen son divididos por él, se refiere al escalamiento de cada dimensión para contener valores en el rango de $[0, 1]$.

Para realizar un escalamiento Min-Max se utiliza la ecuación 2.4 dónde $x^{(i)}$ es una instancia de una dimensión en particular, x_{min} es el valor con la menor magnitud de dicha dimensión, x_{max} es el valor con la mayor magnitud de dicha dimensión y $x^{(i)}_{norm}$ es la instancia normalizada.

$$x^{(i)}_{norm} = \frac{x^{(i)} - x_{min}}{x_{max} - x_{min}} \tag{2.4}$$

La normalización del tipo Min-Max se usa especialmente en aplicaciones que hacen uso de algoritmos ML basados en redes neuronales, ya que dichos algoritmos requieren que sus parámetros estén en un valor entre 0 y 1. Por el hecho de que en éste trabajo de tesis no se hacen uso de redes neuronales, además de los argumentos en el capítulo 2.4.2.2, no se hace uso de la normalización Min-Max.

2.4.2.2. Z-Score

La normalización del tipo Z-score (también llamada estandarización), reescala cada una de las instancias de un vector de forma que se obtengan las propiedades de una distribución normal estándar con $\mu = 0$ y $\sigma = 1$, dónde μ es el valor promedio y σ es la desviación estándar.

Aunque la normalización mediante el escalamiento Min-Max es una técnica comúnmente utilizada la cual es útil cuando se necesitan valores en un intervalo fijo, la estandarización puede ser más practica por múltiples algoritmos de aprendizaje de máquina, especialmente por aquellos algoritmos de optimización. Ésto se debe a que la estandarización mantiene la información de valores inesperados (outliers, en inglés) y hace a los algoritmos menos sensibles a ellos en contraste con el escalamiento Min-Max, la cual escala la información sólo a un rango limitado de valores.

La ecuación para la normalización Z-score se describe en la ecuación 2.5, dónde μ_x es el promedio de la dimensión a analizar, σ_x corresponde a la desviación estándar de dicha dimensión y $x^{(i)}_{std}$ es la instancia estandarizada (normalizada).

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x} \tag{2.5}$$

Por lo antedicho se utiliza el método de normalización Z-score en las bases de datos, NSLKDD y ENSLKDD, de forma que se obtienen bases de datos en un estado adecuado para los algoritmos de aprendizaje de máquina.

Es decir, las dimensiones Nominales (2,3,4,7,12,14,15,21 y 22) las cuales se encuentran con valores del tipo de programación cadena (string, en inglés) se les otorga un valor numérico, con el único objetivo de aplicar Z-score al dataset completo, ya que matemáticamente no tiene sentido la normalización para valores nominales. Además se elimina la dimensión número 20, resultado del análisis de información y entropía. La Figura 2.11 muestra las bases de datos utilizadas en éste trabajo de tesis para el entrenamiento y evaluación de los algoritmos de aprendizaje de máquina en los sistemas IDS. En el caso de la Figura 2.11a describe la base de datos de entrenamiento NSLKDD y la Figura 2.11b muestra la base de datos de evaluación ENSLKDD.

	Duration	ProtocolType	Service	Flag	...	DstHostSerrorRate	DstHostSrvSerrorRate	DstHostRerrorRate	DstHostSrvRerrorRate
0	-0.110262	-0.453014	-0.661298	-0.500424	...	-0.639612	-0.624943	-0.224453	-0.376366
1	-0.110262	2.530010	0.550655	-0.500424	...	-0.639612	-0.624943	-0.387565	-0.376366
2	-0.110262	-0.453014	-0.737045	1.222090	...	1.608515	1.618771	-0.387565	-0.376366
3	-0.110262	-0.453014	-0.434057	-0.500424	...	-0.572168	-0.602506	-0.387565	-0.345061
4	-0.110262	-0.453014	-0.434057	-0.500424	...	-0.639612	-0.624943	-0.387565	-0.376366

(a) Base de datos NSLKDD final utilizada en el entrenamiento de los sistemas IDS

	Duration	ProtocolType	Service	Flag	...	DstHostSerrorRate	DstHostSrvSerrorRate	DstHostRerrorRate	DstHostSrvRerrorRate
0	-0.110262	-0.453014	-0.737045	-1.361682	...	-0.639612	-0.624943	2.874680	2.754098
1	-0.110262	-0.453014	-0.737045	-1.361682	...	-0.639612	-0.624943	2.874680	2.754098
2	-0.109494	-0.453014	-0.661298	-0.500424	...	-0.639612	-0.624943	-0.387565	-0.376366
3	-0.110262	1.038498	-0.585551	-0.500424	...	-0.639612	-0.624943	-0.387565	-0.376366
4	-0.109878	-0.453014	-0.509804	0.360833	...	-0.639612	-0.624943	2.320098	1.846264

(b) Base de datos ENSLKDD final utilizada en la evaluación de los sistemas IDS

Figura 2.11: Bases de datos utilizadas en el entrenamiento de los sistemas IDS

.

2.5. Algoritmos de Aprendizaje de Máquina

En 1959 Artur Samuel [45] tras programar el primer juego de mesa de damas autónomo, define al aprendizaje de máquina cómo el campo de estudio que da a las computadoras la habilidad de aprender sin estar explícitamente programadas. Es decir, sin la necesidad de intervenir con la definición de ninguna sucesión de reglas. Los sistemas de red de detección de intrusos basados en anomalías hacen uso de algoritmos de aprendizaje de máquina (Machine Learning) para clasificar el comportamiento normal de la red de

datos y poder identificar cualquier comportamiento, del cual no se tenga registro con anterioridad que se desvié lo suficiente de la clasificación normal de la red cómo una posible amenaza.

Desde la creación de la base de datos DARPA [38], la cual origino la competencia Knowledge Discovery Database 99 para la evaluación de los sistemas IDS [7] múltiples algoritmos de aprendizaje de máquina han sido utilizados con mejores o menores resultados en su eficacia para detectar anomalías. En [46] se analiza el impacto de los tipos de normalización en la exactitud de los sistemas IDS utilizando la base de datos KDD99, haciendo uso del algoritmo privado de árboles de decisión C5.0, dando resultados muy similares al ganador de la competencia KDD99. En [13] se utiliza el algoritmo SVMs para el entrenamiento y evaluación de los sistemas IDS, ésto a través de la búsqueda heurística aplicando el algoritmo Mutación aleatoria de ascenso de colinas (RMHC) con el propósito de reducir la cantidad de dimensiones en la base de datos, obteniendo bajos tiempos de entrenamiento y evaluación. En [22] se realiza un estudio del análisis de diferentes clasificadores en los sistemas IDS usando la base de datos KDD99, dónde el algoritmo KNN otorga una precisión del 88 % en la detección de anomalías. De lo antedicho se desprende que los algoritmos de aprendizaje de máquina KNN (sección 2.5.4), Árboles de decisión (sección 2.5.9), Bosques aleatorios (sección 2.5.10), y Máquinas de Soporte de Vectores (sección 2.5.5), son utilizados con el objetivo de hacer un análisis de su exactitud en la detección de anomalías de los sistemas IDS.

Para ejemplificar y dicho de otra manera, el principal objetivo de cualquier algoritmo de aprendizaje de máquina utilizado en los sistemas IDS es el descubrir modelos apropiados de la base de datos de datos de entrenamiento para caracterizar/clasificar un comportamiento normal o ataque. El modelo consiguiente es por lo tanto utilizado para realizar predicciones relacionadas a información nunca antes vista, ésto a través de la base de datos de evaluación.

Los algoritmos de aprendizaje de máquina se dividen en tres clases. Algoritmos de aprendizaje supervisado, no supervisado y con técnicas de reforzamiento. La Figura 2.12 describe en forma gráfica su clasificación y componentes principales, mientras que éstos se detallan cómo sigue:

- **Aprendizaje Supervisado:** El principal objetivo del aprendizaje supervisado es aprender de un modelo de entrenamiento *etiquetado*, el cual permita hacer predicciones sobre parámetros nuevos o futuros. El término etiquetado se refiere al hecho de que cada vector en la base de datos de entrenamiento ésta previamente clasificado en alguna clase bien definida. A su vez los algoritmos supervisados se catalogan cómo *clasificadores*, cuando predicen una salida discreta categórica, o cómo *algoritmos de regresión*, dónde predicen un valor continuo.

- **Aprendizaje No Supervisado:** En el aprendizaje no supervisado *no se conoce* con anterioridad ningún tipo de patrón o etiqueta cuanto se utiliza una base de datos de entrenamiento, es decir, a partir de información completamente desconocida se consigue una estructura. Usando técnicas de aprendizaje no supervisado es posible explorar la estructura de la información otorgada para extraer información con algún significado sin ningún tipo de guía, cómo el etiquetado en el aprendizaje supervisado o las funciones de recompensa en el aprendizaje con reforzamiento.
- **Aprendizaje con reforzamiento:** El objetivo es desarrollar un sistema (llamado *agente*) el cual mejora su rendimiento a través de interacciones con el ambiente. Ya que la información sobre el estado actual del entorno incluye típicamente una señal llamada *recompensa* (reward signal, en inglés), se trata al aprendizaje con reforzamiento cómo un campo relacionado al aprendizaje supervisado; sin embargo, en el aprendizaje con reforzamiento éste tipo de retroalimentación del entorno *no se refiere directamente* a una clase de etiquetado, sino a una medida de qué tan bien la acción realizada es evaluada por la función reward, por lo tanto a través de su interacción con el ambiente un agente puede usar el aprendizaje con reforzamiento para aprender la serie de acciones que maximiza la función reward, ésto mediante prueba y error.

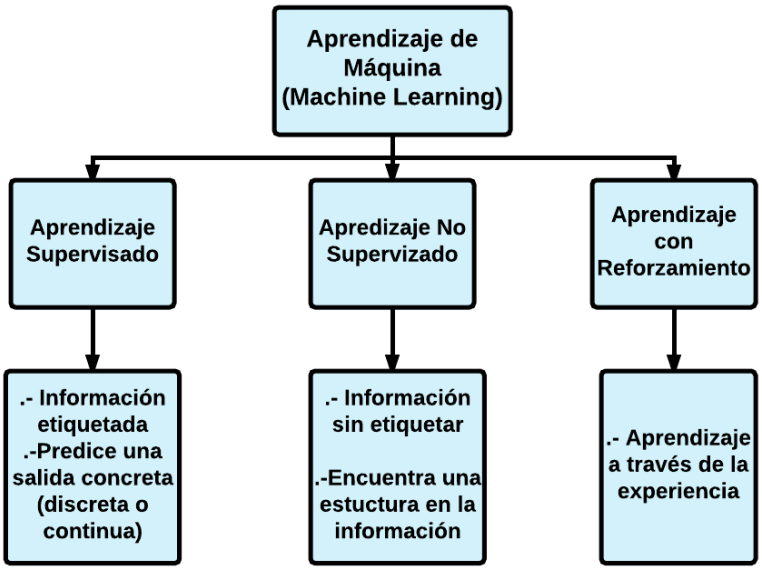
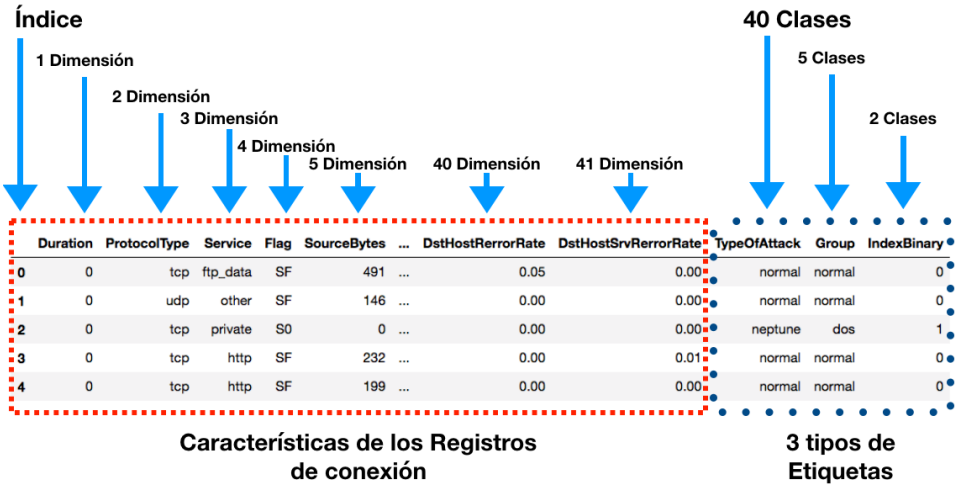


Figura 2.12: Clasificación de los algoritmos de aprendizaje de máquina

Tanto la base de datos NSLKDD y ENSLKDD utilizadas en éste trabajo de tesis para el análisis de la exactitud de los sistemas IDS cuentan con un etiquetado en cinco clases principales, así cómo el total de los 40 tipos de ataques en ambos dataset. La Figura 2.10 en la página 34 muestra en la dimensión número 42 llamada *TypeOfAttack* la clasificación de los 40 tipos de ataques mientras que la dimensión número 43 llamada *Group* describe la clasificación en alguna de las cinco clases principales, DoS, R2L, U2R, Normal y Probe. De lo antedicho se desprende que al estar cada uno de los registros de conexión en NSLKDD y ENSLKDD, etiquetados en alguna clase, en éste trabajo de tesis se utiliza el tipo de aprendizaje supervisado del cual se ahonda en la sección 2.5.1.

2.5.1. Aprendizaje Supervisado en los sistemas IDS

De lo antedicho un sistema IDS basado en anomalías el cual usa técnicas de aprendizaje de máquina mediante el aprendizaje supervisado requiere de una base de datos etiquetada, cómo es el caso de NSLKDD y ENSLKDD. La Figura 2.13 muestra a detalle las primeras cinco instancias de la base de datos NSLKDD dónde la parte izquierda corresponde a cada registro de conexión, mientras que el lado derecho representa los tres tipos de etiquetas para cada clase. El etiquetado de 40 clases corresponde a todos los ataques contenidos tanto en NSLKDD cómo en ENSLKDD (tabla 2.5, de la página 33), las 5 clases se refiere a las cinco clases principales DoS, R2L, U2R, R2L y Probe; las últimas 2 clases se refiere a la clasificación binaria, dónde un registro de conexión es considerado cómo ataque o no.



	Duration	ProtocolType	Service	Flag	SourceBytes	...	DstHostRerrorRate	DstHostSrvRerrorRate	TypeOfAttack	Group	IndexBinary
0	0	tcp	ftp_data	SF	491	...	0.05	0.00	normal	normal	0
1	0	udp	other	SF	146	...	0.00	0.00	normal	normal	0
2	0	tcp	private	S0	0	...	0.00	0.00	neptune	dos	1
3	0	tcp	http	SF	232	...	0.00	0.01	normal	normal	0
4	0	tcp	http	SF	199	...	0.00	0.00	normal	normal	0

Figura 2.13: Aprendizaje supervisado en NSLKDD

El siguiente proceso describe de manera explicita la elaboración de los

sistemas IDS en éste trabajo de tesis. Los sistemas IDS pasan por una parte de preprocesado en la cual se realiza una normalización (sección 2.4.2.2) de NSLKDD y ENSLKDD, después la base de datos NSLKDD se utiliza para el entrenamiento del algoritmo de aprendizaje de máquina, ésta fase se llama de *aprendizaje*. La base de datos de entrenamiento y el algoritmo de aprendizaje de máquina generan un *modelo predictivo*, a través del cual puede realizarse una evaluación del sistema utilizando la base de datos ENSLKDD junto con sus correspondientes etiquetas.

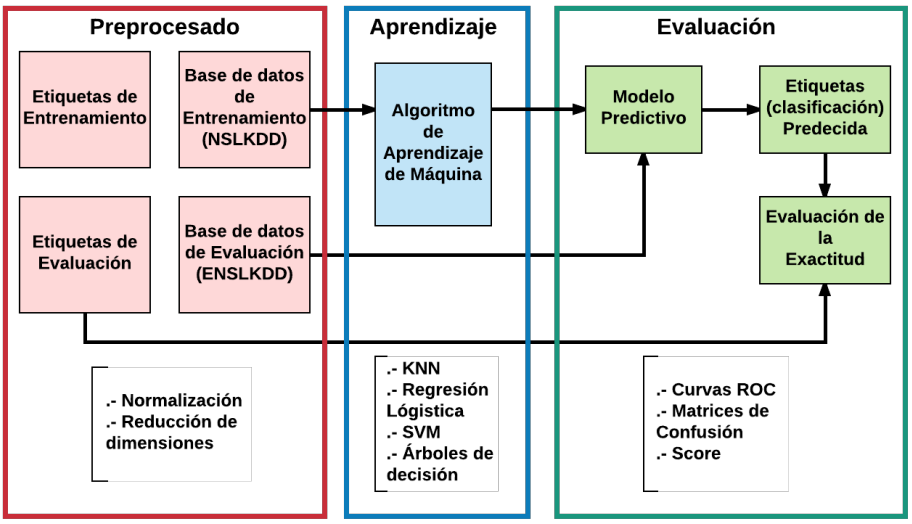


Figura 2.14: Proceso de creación de los sistemas IDS

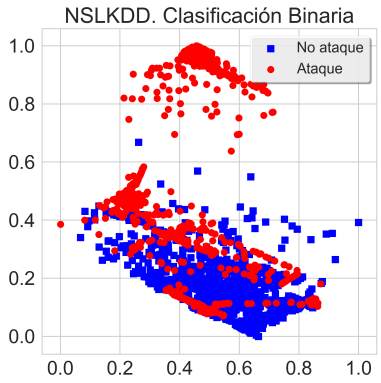
La Figura 2.14 muestra de forma gráfica el proceso de elaboración de los sistemas IDS, dónde la parte de la izquierda corresponde al preprocesado con la normalización y reducción de dimensiones (métodos de tipo filtro o envoltura), la parte central pertenece a la implementación del algoritmo de aprendizaje de máquina y la parte de la derecha a los tipos de evaluación una vez que el algoritmo de aprendizaje haya sido entrenado.

2.5.2. Visualización de NSLKDD y ENSLKDD

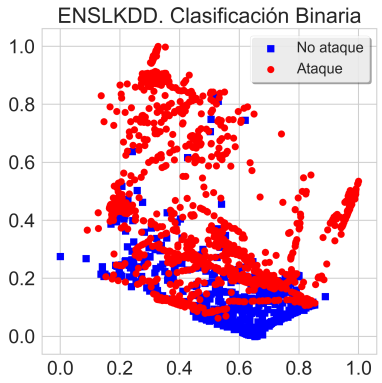
Las bases de datos NSLKDD y ENSLKDD cuentan con un total de 40 dimensiones, para poder visualizar la distribución de cada uno de los ataques que ella contiene es necesario utilizar algún algoritmo de reducción de dimensiones para llevarla a un plano 2 o 3 dimensional. Los algoritmos de reducción de dimensiones no sólo son utilizados con el objetivo de visualizar la distribución de alguna base de datos, sino, además, cómo métodos de tipo filtro los cuales convergen mucho más rápido que los métodos del tipo

envoltura. El objetivo de éste trabajo de tesis es utilizar algoritmos del tipo envoltura a través de la búsqueda heurística, por lo antedicho no se detalla la implementación y desarrollo de los algoritmos utilizados con el objetivo de visualización, Análisis por Componentes Principales (PCA), y t-SNE; sin embargo el autor recomienda [47] para un mayor entendimiento del PCA y [48] dónde Laurens presenta su algoritmo t-SNE. Los algoritmos PCA y t-SNE son implementados mediante las clases *sklearn.decomposition.PCA* [49] y *sklearn.manifold.TSNE* [50], embebidas en las librerías de código abierto de *ScikitLearn* [51]. Al contar con más de 100000 y 20000 instancias para el caso de NSLKDD y ENSLKDD respectivamente, se selecciona una porción aleatoria de 3500 instancias para su visualización a través de un *seed=8* con el fin de poder replicar los resultados aquí obtenidos en las librerías de *Numpy* [52]. El apéndice A.3 muestra el código completo de su implementación en el lenguaje Python.

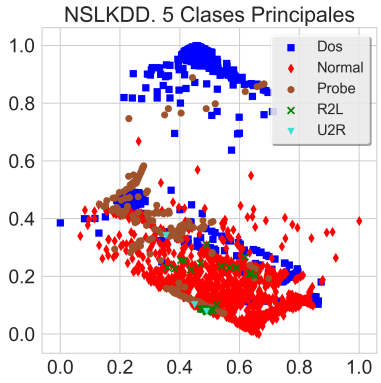
Haciendo uso del algoritmo PCA se consigue reducir a un total de dos dimensiones ambas bases de datos a través del primer y segundo componente principal con un tiempo de cómputo del algoritmo de $0.12536s$. Ésta reducción de dimensiones a través de PCA se muestra en la Figura 2.15, dónde las Figuras 2.15a y 2.15b muestran la clasificación cuando un registro de conexión es considerado ataque o no ataque, mientras que las Figuras 2.15c y 2.15d muestran la clasificación en las clases DoS, R2L, U2R, Normal y Probe. El algoritmo PCA ofrece una de las más rápidas convergencias en los algoritmos de reducción de dimensiones al hacer uso de eigenvectores y eigenvalores, sin embargo no ofrece la mejor visualización de las múltiples clases, cómo se puede observar en la Figura 2.15c y 2.15d, dónde no es posible apreciar una clara segmentación en los registros de conexión de las clases R2L, DoS, Normal los cuales terminan agrupados en la parte inferior.



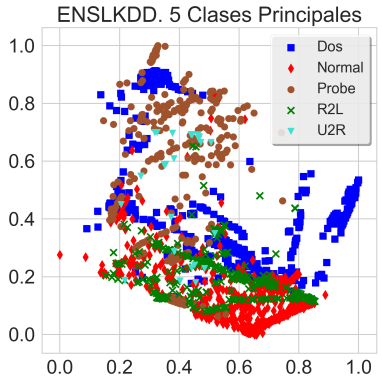
(a) Distribución de la clasificación binaria en NSLKDD



(b) Distribución de la clasificación binaria en ENSLKDD



(c) Distribución de la clasificación en cinco clases en NSLKDD



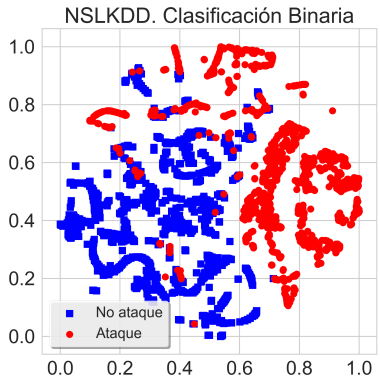
(d) Distribución de la clasificación en cinco clases en ENSLKDD

Figura 2.15: Visualización de la distribución de la clasificación binaria y cinco clases en NSLKDD y ENSLKDD utilizando el algoritmo PCA

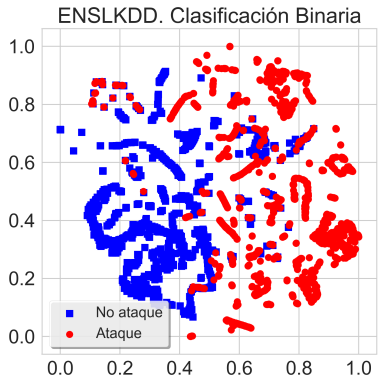
.

La utilización del algoritmo PCA en la Figura 2.15 es con el objeto de mostrar las diferencias del tiempo de convergencia y calidad de visualización ante el algoritmo t-SNE el cual pese a ser computacionalmente más demandante es una técnica para la reducción de dimensiones particularmente adecuada para la visualización de bases de datos de altas dimensiones. t-SNE es una variación del algoritmo Stochastic Neighbor Embedding, el cual produce mejores visualizaciones a través de la reducción de la tendencia a aglomerar las múltiples clases de las instancias en la base de datos en el centro del mapa. Laurens en [53] ofrece diferentes implementaciones de su algoritmo para los lenguajes de programación Python, C++, Matlab, R, en-

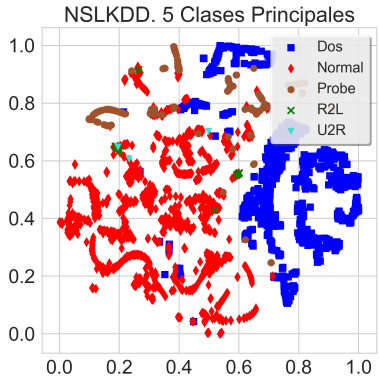
tre otros. Haciendo uso del algoritmo t-SNE en NSLKDD y ENSLKDD para su reducción a dos dimensiones se obtiene la Figura 2.16, con un tiempo de cómputo de $69.230106s$. Dónde de forma análoga a la Figura 2.15, las Figuras 2.16a y 2.16b muestran la clasificación cuando un registro de conexión es considerado ataque o no ataque, mientras que las Figuras 2.16c y 2.16d muestran la clasificación en las clases DoS, R2L, U2R, Normal y Probe



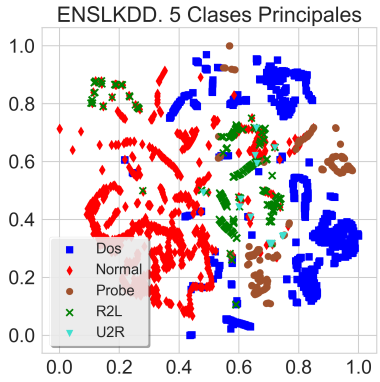
(a) Distribución de la clasificación binaria en NSLKDD



(b) Distribución de la clasificación binaria en ENSLKDD



(c) Distribución de la clasificación en cinco clases en NSLKDD



(d) Distribución de la clasificación en cinco clases en ENSLKDD

Figura 2.16: Visualización de la distribución de la clasificación binaria y cinco clases en NSLKDD y ENSLKDD utilizando el algoritmo t-SNE

.

Es posible observar mediante las Figuras 2.15 y 2.16 que no es trivial la clasificación de los algoritmos de aprendizaje de máquina en los sistemas IDS y la distribución de los registros de conexión demuestran que se posee

un alto nivel de complejidad, por lo tanto *no es viable* realizar un método de clasificación lineal debido a la forma en que se concentran cada una de las cinco clases, la aproximación que se debe realizar con los clasificadores en los algoritmos de aprendizaje de máquina es a través de una aproximación no paramétrica ya que no se asume ninguna forma de distribución de probabilidad de cada una de las clases y por consiguiente, es posible conseguir un alto grado en la exactitud de su clasificación pese a no tener una distribución lineal. Los algoritmos de aprendizajes de máquina no paramétricos más utilizados son K-Vecinos Cercanos (KNN), Árboles de Decisión (DT), Bosques Aleatorios (RF) y Máquinas de Soporte de Vectores (SVM's), algoritmos de aprendizaje de máquina utilizados en éste trabajo de tesis y descritos a continuación.

2.5.3. Algoritmos de aprendizaje de máquina paramétricos y no paramétricos

La rama de aprendizaje de máquina puede ser resumido a muy alto nivel cómo una función f que se encarga de mapear variables de entrada X compuesta de una base de datos de evaluación, a una serie de variables de salida Y, es decir, $Y = f(X)$, dónde los diferentes tipos de algoritmos *aprenden* la función de mapeo mediante una base de datos de entrenamiento. La forma de la función f es previamente desconocida, y se desprende que diferentes algoritmos hacen diferentes suposiciones o poseen ciertas tendencias sobre la forma de f y cómo puede ser aprendida. En consecuencia a éstas suposiciones se originan dos clases de algoritmos de aprendizaje de máquina, del tipo paramétrico y no paramétrico.

2.5.3.1. Algoritmos de aprendizaje de máquina paramétricos

Tener una forma de suposición de la distribución de las múltiples clases en la base de datos de entrenamiento puede ayudar a simplificar enormemente el proceso de aprendizaje, sin embargo ésto puede también *limitar* qué puede ser aprendido. Éste tipo de algoritmos que simplifican la función f a una forma conocida son llamados algoritmos de aprendizaje del tipo paramétrico. Los algoritmos del tipo paramétricos involucran una aproximación de su modelado basado en dos pasos [54]:

1. Se empieza realizando una suposición sobre la forma de f.
   Con fines demostrativos y mediante la suposición de un modelo lineal, f es descrita mediante la forma de regresión lineal en la ecuación 2.6

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \tag{2.6}$$

2. Después de que un modelo de f ha sido seleccionado se utiliza un procedimiento el cual utilice la base de datos de entrenamiento para

entrenar el modelo.
En el caso de la ecuación 2.6 se necesita estimar los parámetros β_0, β_1 y β_2. La aproximación más común para entrenar el modelo, es decir, encontrar los parámetros β, es a través de mínimos cuadrados, sin embargo existen múltiples métodos para encontrar dichos parámetros.

Los algoritmos de aprendizaje paramétricos reducen el problema de estimar la función f a estimar únicamente un conjunto reducido de parámetros. La principal desventaja de la aproximación paramétrica es que el modelo utilizado usualmente no coincide con la verdadera forma de f al no ser previamente conocida. Los métodos paramétricos convergen mucho más rápido que los no paramétricos y aprenden mucho más rápido de la base de datos de entrenamiento, sin embargo al elegir una forma de f se encuentran restringidos a dicha forma, éstos métodos son sólo adecuados para bases de datos de una complejidad limitada y no se ajustan a las necesidades del análisis de la exactitud de los algoritmos de aprendizaje de máquina en éste trabajo de tesis.

2.5.3.2. Algoritmos de aprendizaje de máquina no paramétricos

Los algoritmos que no hacen fuertes suposiciones sobre la forma de la función de mapeo f son llamados algoritmos de aprendizaje de máquina no paramétricos; al no tener ninguna suposición a priori son libres de aprender cualquier forma de la función f mediante la base de datos de entrenamiento. Los algoritmos de aprendizaje de máquina no paramétricos buscan un estimado de f el cual se acerque lo más posible a los puntos de la base de datos de entrenamiento sin ser demasiado severo ni moderado; son adecuados cuando se tiene una amplia cantidad de información en la base de datos de entrenamiento, ningún conocimiento de f y no se desea concentrarse demasiado en encontrar las características adecuadas de configuración de los algoritmos [55]. Éstos algoritmos poseen flexibilidad al ser capaces de aproximar múltiples funciones de f, son capaces de ser adaptados a problemas con una alta complejidad al no tener suposiciones a priori y generalmente resultan en un mayor rendimiento en su exactitud de clasificación, sin embargo, requieren mucho más instancias en la base de datos de entrenamiento para tener una buena evaluación de la función f, al depender su cantidad de parámetros del tamaño de la base de datos de entrenamiento necesitan un mayor tiempo computacional, además tienen un mayor riesgo de sobreentenar el algoritmo de aprendizaje de máquina para adecuarse únicamente a la base de datos de entrenamiento perdiendo la habilidad de generalizar.

Los algoritmos no paramétricos sufren de una desventaja mayor, ya que al no reducir el problema de estimar a f a un pequeño número de parámetros, un gran número de instancias (mucho más de las necesarias típicamente con una aproximación paramétrica) son requeridas para obtener un buen

estimado para la función f. Sin embargo al tener las bases de datos utilizadas para el análisis de la exactitud de los sistemas IDS, NSLKDD y ENSLKDD, 125973 y 22543 instancias respectivamente, su principal problemática de los algoritmos no paramétricos se muestra solventada.

2.5.4. K-Nearest Neighbors

K-Nearest Neighbors (KNN), en español, K vecinos cercanos, es un algoritmo del tipo clasificador de la rama de aprendizaje de máquina no paramétrico, el cual no hace ninguna suposición de la forma de f. Dado un valor entero positivo de K denominado cómo el número de vecinos a considerar, una instancia de evaluación χ_i, definido de un conjunto de instancias de evaluación $\chi \in \Re^{n \times p}$, dónde n se refiere al total de instancias y p al total de características (dimensiones) y χ es también llamada base de datos de evaluación, el clasificador KNN identifica los K puntos en la base de datos de entrenamiento X los cuales se encuentran más cercanos a χ_0, representados por η_0. A continuación KNN evalúa la probabilidad condicional para la clase j cómo la fracción de puntos en η_0 los cuales pertenecen a dicha clase, cómo se describe en la ecuación 2.7

$$Pr\left(Y = j | \chi = \chi_0\right) = \frac{1}{K} \sum_{i \in \eta_0} I\left(y_i = j\right) \tag{2.7}$$

Dónde Y se refiere al conjunto total de clases en la base de datos de entrenamiento y evaluación (comúnmente se tienen la misma cantidad de clases); y_i se refiere a una clase particular de Y, y j se refiere a la clase perteneciente a dicho vecino, en otras palabras, KNN aplica la regla de bayes para clasificar el punto de observación x_0 a aquella clase con la mayor probabilidad dentro de la cantidad de vecinos η_0.

A través de [56] es posible obtener la Figura 2.17, la cual muestra un ejemplo de una distribución de una base de datos de 26 elementos con dos clases generada con puntos aleatorios en el plano, denominada R; 13 de los elementos en R pertenecen a la primer clase representada a través del símbolo $\bullet$, mientras que los 13 elementos restantes en R pertenecen a la segunda clase representada por el símbolo $\triangle$. R cuenta con dos características, es decir, dos dimensiones.

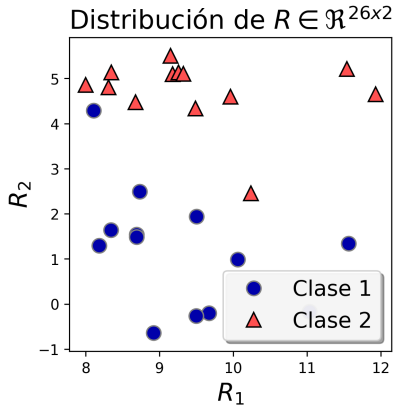
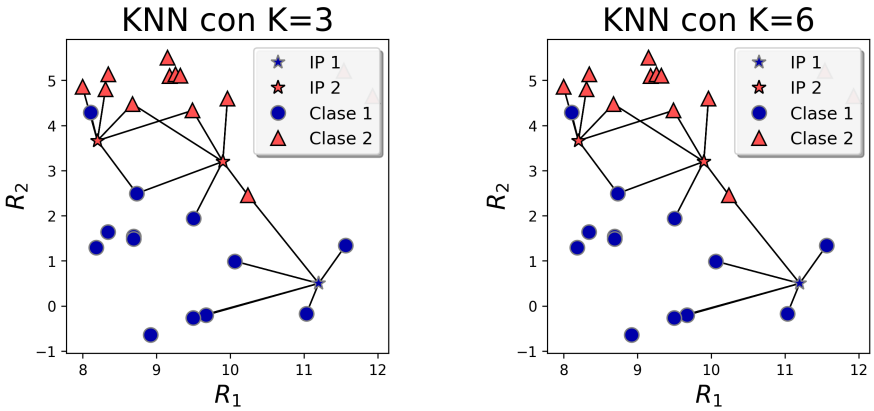


Figura 2.17: Distribución de la base de datos de prueba R

La Figura 2.18a muestra el algoritmo KNN aplicado a la base de datos R de la Figura 2.17, con tres instancias de prueba (evaluación). $IP1$ es una instancia de evaluación perteneciente a la primer clase mientras que $IP2$ es una instancia de evaluación perteneciente a la segunda clase. Es posible observar que cada una de las instancias de evaluación buscan el total de $k = 3$ vecinos cercanos y, una vez encontrados dichos vecinos, KNN contempla la probabilidad de que una instancia x_0 pertenezca a la y_i clase a través del total de elementos en η_0. Paralelamente la Figura 2.18b considera un valor de $k = 6$ vecinos, dónde $IP1$ y $IP2$ evalúan los 6 vecinos más cercanos a ellos para su clasificación.



(a) Algoritmo KNN aplicado en R con K=3

(b) Algoritmo KNN aplicado en R con K=6

Figura 2.18: Algoritmo KNN aplicado en R con valores de K=3,6

.

El procedimiento general del algoritmo KNN en descrito en pseudocódigo mediante el algoritmo 2.1 el cual permite visualizar su fácil implementación, no obstante, pese a ésto, el algoritmo de clasificación KNN ofrece muy buenos resultados en su exactitud. Incluso podría considerarse el análogo al método de ascenso de colinas por mutación aleatoria en la búsqueda heurística, el cual suele sobresalir en resultados de minimización y maximización pese a tener una de las implementaciones más sencillas.

Algoritmo 2.1 Clasificación mediante K-Nearest Neighbors

```
1: Xtra[]=Base de datos de entrenamiento
2: Xtest[]=Base de datos de evaluación
3: Y[]=Etiquetas correspondientes a las clases en Xtra
4: K=Cantidad de vecinos
5: d_i[]=Distancias correspondientes a Xtest_i y Xtra_i
6: for Xtest_i to length(Xtest) do
7:     for Xtra_i to length(Xtra) do
8:         d_i[]=CalculoDistancia(Xtest_i,Xtra_i)
9:     end for
10:    Indice[]=IndiceMinKValores(d_i)
11:    ClasesEnKVecinos[]=Y[Indice]
12:    ClasePredecidaXtest_i[]=ProbabilidadBayes(ClasesEnKVecinos)
13: end for
```

Del algoritmo 2.1 se observa que los parámetros de configuración de KNN se reducen al total de vecinos cercanos k y al algoritmo para el calculo de las distancias entre un punto de prueba con toda la base de datos de entrenamiento. El escoger un valor adecuado de k resulta por lo antedicho, crítico. Un valor elevado de k provocará que el clasificador se sobreentrene, es decir, al probar el clasificador KNN entrenado y probado con la misma base de datos de entrenamiento dará cómo resultado una exactitud de clasificación de casi el 100 %, sin embargo, al ser evaluado con elementos los cuales no se encuentren en la base de datos de entrenamiento (desconocidos) mermará de forma considerable su exactitud al no poseer el clasificador la suficiente generalización. Por el contrario, un valor muy pequeño de k provocará que el clasificador sea sumamente flexible y no sea capaz de encontrar patrones pertenecientes a la base de datos de evaluación. El valor de k utilizado en éste trabajo de tesis corresponde a $k = 10$, además del método de cálculo de distancias es a través del algoritmo de *minkowski*. Ambos parámetros son obtenidos mediante una búsqueda de GridSearch [57] y haciendo uso de las bases de datos para los sistemas IDS, ENSLKDD y NSLKDD.

2.5.5. Support Vector Machines

El algoritmo de aprendizaje de máquina Support Vector Machines (SVMs), es un algoritmo clasificador del tipo *no paramétrico* al poseer la aplicación de algún tipo de kernel K; es la generalización de los algoritmos, Maximal Margin Classifier (MMC), el cual requiere que las clases binarias que contiene sean linealmente separables, y del algoritmo Support Vector Classifier (SVC) el cual realiza una definición del hiperplano de manera *suave*, contrariamente a MMC el cual es más estricto en su definición. Comúnmente y, es necesario mencionar, en parte de la literatura son llamados de manera errónea, al algoritmo Maximal Margin Classifier cómo Support Vector Classifier y al algoritmo Support Vector Classifier cómo Support Vector Machines. Por lo antedicho y con el objeto de definir justamente el algoritmo SVMs, se aborda desde MMC dónde ambos están basados en hiperplanos.

Un hiperplano es un espacio p-dimensional afín (es decir, el subespacio *no necesita* pasar por el origen) de dimensiones $p-1$, en el cual en un conjunto de datos de entrenamiento (también llamado base de datos de entrenamiento) $X \in \Re^{n \times p}$, dónde n se refiere a la cantidad de instancias y p a la cantidad de dimensiones/características, la definición matemática del hiperplano para el caso específico de dos dimensiones se expresa mediante la ecuación 2.8

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0 \tag{2.8}$$

De 2.8 es posible observar que el hiperplano es una linea, dónde los parámetros β son escalares que moldean al hiperplano y se dice que éste queda definido para cualquier valor de $X = (X_1, X_2)^T$ que cumpla con 2.8, es decir cualquier valor de X que cumpla a 2.8 se encuentra en el hiperplano. Éste concepto de hiperplano se puede extender para p dimensiones cómo muestra la ecuación 2.9

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0 \tag{2.9}$$

En caso de un valor de X el cual no cumpla con 2.9, y en su lugar resulta cumplir con la expresión 2.10, significa que X reside en un lado del hiperplano.

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p > 0 \tag{2.10}$$

Para el caso contrario cuando con un valor de X se resulta cumplir con la condición 2.11, significa que el valor de X reside en el lado contrario a 2.10

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p < 0 \tag{2.11}$$

Dicho de otra manera, en una base de datos de evaluación $\chi \in \Re^{n \times p}$, es posible obtener la clasificación de todo el conjunto χ o de una instancia particular χ_n, obteniendo el hiperplano correspondiente a la separación de sus dos clases y conociendo el signo del lado izquierdo de la ecuación 2.9.

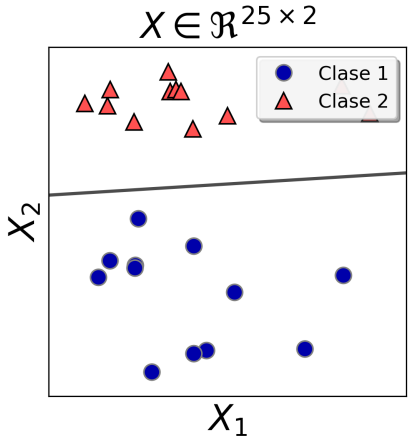


Figura 2.19: Base de datos X con $p = 2$ e hiperplano de clasificación

Lo antedicho se puede observar mediante la Figura 2.19, dónde la mayoría de elementos de la primer clase en X se encuentran en la parte inferior mientras que la mayoría de elementos de la segunda clase se encuentran en la parte superior y ambos están separados por un hiperplano. Sin embargo, cómo muestra la Figura 2.20, la cantidad de hiperplanos posibles que separen de manera adecuada las dos clases en X es infinita. Para reducir y optimizar la aproximación del hiperplano que otorgue la mejor clasificación se utiliza el método *Maximal Margin Classifier*

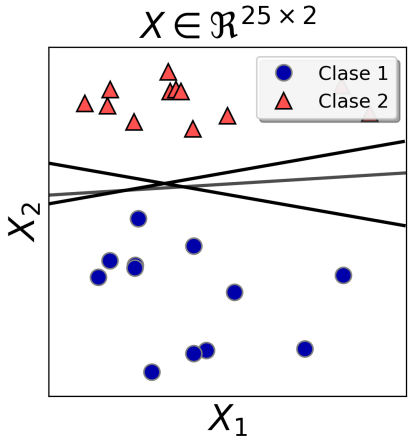


Figura 2.20: Múltiples hiperplanos de clasificación para X

2.5.6. Maximal Margin Classifier

De forma general si las instancias x_n de X pueden ser *completamente separables* utilizando un hiperplano, existe en consecuencia una infinita cantidad de hiperplanos capaces de realizar ésta separación. Para solucionar ésta problemática se utiliza el algoritmo Maximal Margin Classifier (Clasificador de Máximo Margen, en español), el cual tiene cómo objetivo encontrar el hiperplano llamado *Maximal Margin Hyperplane (MMH)*, el cual es el hiperplano que se encuentra más lejano de los puntos de entrenamiento x_n, es decir, es necesario calcular la distancia perpendicular de cada punto de observación a un hiperplano en particular, entre menor sea dicha distancia ésta se convertirá en la mínima distancia de los puntos x_n al hiperplano, a ésta mínima distancia se le llama margen; de lo antedicho se desprende que el hiperplano a encontrar MMH, sea *aquel hiperplano que encuentre la máxima-mínima distancia a los puntos* x_n, ésto se puede apreciar en la Figura 2.21, dónde los vectores que tocan los extremos del margen son llamados vectores de soporte, y el hiperplano MMH es aquel que otorga la mejor clasificación.

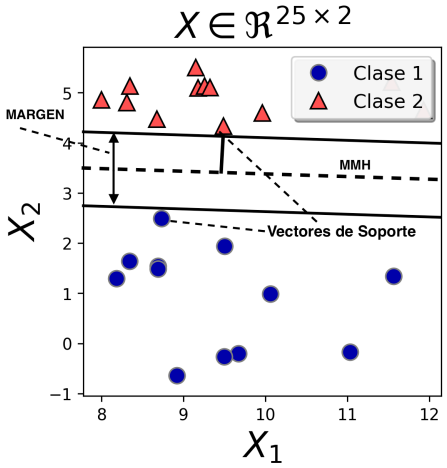


Figura 2.21: Maximal Margin Classifier

El clasificador MMC puede ser utilizado *siempre que exista* un hiperplano MMH el cual *sea capaz* de separar *ambas clases completamente* en X, éste no puede ser utilizado si existen elementos x_n los cuales se encuentren muy contiguos a la clase 1 si ellos pertenecen a la clase 2, ya que debe existir algún tipo de penalización en la clasificación de elementos χ_n. Por lo tanto se desprende que exista una generalización para MMC la cual sea capaz de crear un hiperplano MMH, para situaciones dónde las clases no se encuentran *bien definidas*, ésta generalización es llamada *Support Vector Classifiers*.

2.5.7. Support Vector Classifiers

MMC es una de las primeras aproximaciones para un problema con el cual realizar una clasificación binaria siempre que *exista* un hiperplano el cual sea capaz de separar ambas clases . Sin embargo, cómo muestra la Figura 2.22 no siempre puede existir un hiperplano que separe de forma exacta ambas clases, a diferencia de MMC en la Figura 2.20 se agregaron dos instancias x_n, para un total de $n = 27$, dónde una de ellas se encuentra muy cercana a la clase 1, mientras que la siguiente se encuentra muy cercana a la clase 2.

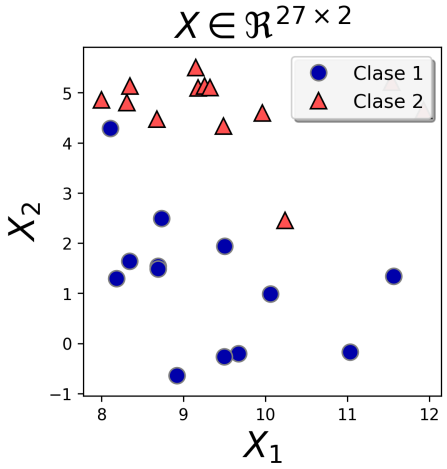


Figura 2.22: Base de datos X, con clases no separables mediante MMC

El concepto de MMC se puede extender por tanto, a situaciones cómo la de la Figura 2.22 con el objetivo de encontrar un hiperplano que si bien no sea capaz de separar de forma exacta ambas clases, puede *aproximarse*, utilizando un *margen suave*, llamado así porque el margen puede contener instancias x_n dentro de él, a diferencia de MMC. La generalización de MMC es llamada Support Vector Classifiers (Clasificadores de soporte de vectores, en español). SVC funciona con la idea principal de realizar una clasificación correcta de la *mayoría* de las instancias de X, y asume la clasificación errónea de ciertas instancias con el objetivo de obtener una mejor exactitud con las instancias restantes. Por lo tanto SVC clasifica una instancia de evaluación χ_n dependiendo de qué lado del hiperplano se encuentra. Éste hiperplano se selecciona de forma que sea posible separar la mayoría de las instancias en X en dos clases aún cuando cometa algunos errores, dicho hiperplano se encuentra a través de la solución del problema de optimización de las ecuaciones 2.12- 2.15.

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{maximizar} \; M \tag{2.12}$$

$$Sujeto \quad a \quad \sum_{j=1}^{p} \beta_j = 1 \tag{2.13}$$

$$y_i \left(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}\right) \geq M \left(1 - \epsilon_i\right) \tag{2.14}$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C \tag{2.15}$$

Dónde C es un parámetro de modificación no negativo, $y_1, ..., y_n \in \{-1, 1\}$ dónde $y_i = 1$ representa la primer clase, $y_i = -1$ representa la segunda clase correspondiente a la instancia χ_i, M es el ancho del margen y se busca que ésta cantidad sea lo más grande posible, $\epsilon_1, ..., \epsilon_n$ son variables escalares llamadas slack las cuales permiten que las instancias x_n se encuentren en un punto erróneo del hiperplano. Una vez que se han resuelto las ecuaciones 2.12- 2.15, es posible clasificar una instancia de evaluación χ_n determinando en qué lado del hiperplano reside. La variable ϵ_i expresa dónde la i-ésima observación está ubicada en relación al hiperplano y al margen, es decir, si $\epsilon_i = 0$ entonces la i-ésima observación se encuentra en el lado correcto del margen, si $\epsilon_i > 0$ se encuentra en el lado contrario del margen y si $\epsilon_i > 1$ se encuentra en el otro lado del hiperplano. Cabe destacar que para una correcta interpretación de margen e hiperplano se debe observar de forma detenida la Figura 2.21. El parámetro C es considerado un parámetro que permite controlar si el sistema está sobre entrenado o poco entrenado, a través de éste parámetro es posible establecer una cantidad de instancias χ_n las cuales se encuentren entre algún punto del margen y el hiperplano, en otras palabras, permite establecer una cantidad de instancias las cuales transgredan éste margen e hiperplano.

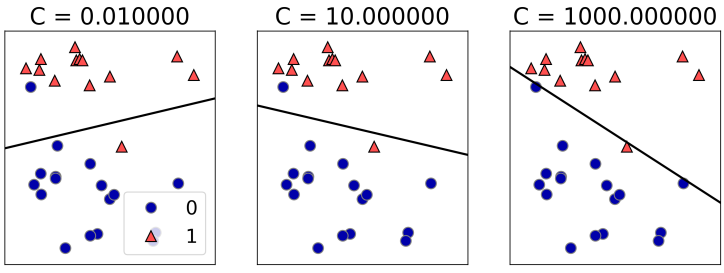


Figura 2.23: Clasificador SVC con Valores de C=0.01,10,1000

Mientras el parámetro C crezca el sistema SVC se volverá más tolerante a las trasgresiones al margen, en consecuencia el margen será más grande, en caso contrario con valores pequeños de C el sistema se volverá menos

tolerante a transgresiones al margen y éste será más pequeño, lo antedicho se puede observar en la Figura 2.23, dónde con $C = 0.01$ el resultado del hyperplano se aproxima al obtenido mediante MMC y cuando $C = 1000$ existen instancias las cuales residen en el propio hiperplano.

Los algoritmos clasificadores de aprendizaje de máquina SVC y MMC son utilizados en aquellos casos dónde exista un hiperplano que si bien puede no separar de forma exacta las instancias en la base de datos de entrenamiento para el caso de SVC, sí la mayoría de ellas. Sin embargo existen situaciones cómo la presentada en los sistemas IDS en la base de datos NSLKDD y ENSLKDD, cuya visualización se puede observar en la Figura 2.16 dónde no existe ningún tipo de hiperplano capaz de realizar alguna clasificación, para ello se utilizan los sistemas *Support Vector Machines* los cuales permiten hacer uso de la técnica llamada *Kernel Trick*, con la cual es posible realizar algún tipo de clasificación *No lineal, no paramétrica.*

2.5.8. Support Vector Machines y Kernel Trick

El algoritmo Support Vector Machines (SVMs), mediante la técnica del Kernel Trick (Truco de Kernel, en español), permite solventar la problemática de los algoritmos MMC y SVC los cuales son incompetentes para una clasificación no lineal cómo la de la Figura 2.16. SVMs parte de la idea de que en una base de datos X no separable de forma lineal mediante un hiperplano, puede ser aplicada una transformación ϕ tal que $\phi : \Re^p \rightarrow \Re^h$, dónde $h > p$, en la cual *exista* un hiperplano β el cual se capaz de separar ambas clases. La Figura 2.24 muestra un conjunto de datos X los cuales *no pueden ser linealmente separables*, por lo tanto es aplicada una transformación $\phi : \Re^2 \rightarrow \Re^3$, llevando la base de datos X de 2 dimensiones a un plano de 3 dimensiones, en el cual pueda existir un hiperplano β capaz de realizar ésta separación.

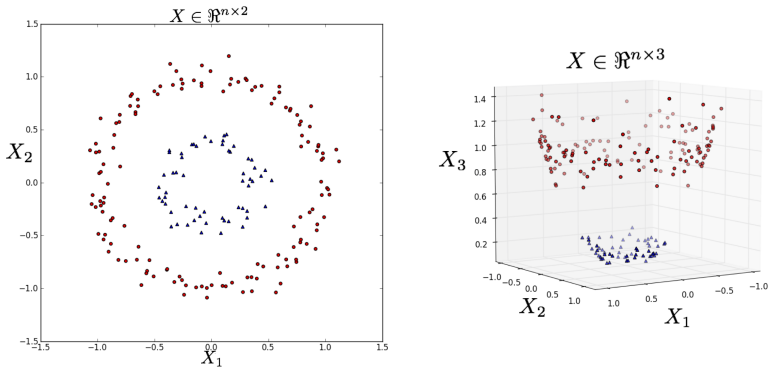


Figura 2.24: Base de datos X en dos dimensiones y tres dimensiones tras la transformación ϕ

De la Figura 2.24 es posible observar que existe un hiperplano el cual es capaz de separar ambas clases cuando éstas se proyectan a una tercera dimensión, en consecuencia para obtener el clasificador f_{svm} es necesario transformar la base de datos de entrenamiento X a X' mediante ϕ, seguido es necesario entrenar un clasificador lineal SVC con X' con el cual obtener el hiperplano de clasificación. Para la clasificación de un elemento de prueba χ_n, es necesario que ésta instancia sea primero transformada mediante ϕ, es decir, $\chi'_n = \phi(\chi_n)$, de forma que su clase es obtenida mediante $f_{svm}(\chi'_n)$. La Figura 2.25 muestra la proyección de dos a tres dimensiones que se realizó en la Figura 2.24, con el hiperplano β el cual separa ambas clases perfectamente; la Figura de su derecha muestra el hiperplano β proyectado de la tercera dimensión a la segunda, dando cómo consecuencia una clasificación *no lineal*.

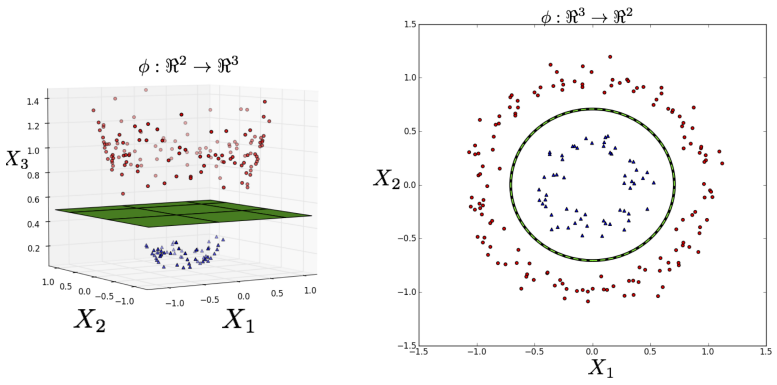


Figura 2.25: Hiperplano β en $p = 2$ y su proyección con $p = 3$

La idea de extender el espacio de una dimensión p a una dimensión mayor h permite generalizar MMC y SVC a una clasificación *no lineal*, sin embargo, ésta generalización a través de la transformación ϕ tiene la consecuencia de requerir mayor demanda computacional al extender el espacio de $\Re^p$ a $\Re^h$ dónde $h > p$, si h crece de forma que resulta mucho mayor a p requerirá en consecuencia mucho mayor demanda de cómputo y memoria. Para el caso de una función ϕ polinomial ϕ_p de segundo grado usado a menudo en SVMs, aplicado a un dataset X de dos dimensiones, implicará realizar la transformación de manera que resulta $\phi[x_1, x_2] = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2c}x_1, \sqrt{2c}x_2, c]$ [58], es decir, ésta única transformación agrega tres dimensiones adicionales $\Re^2 \to \Re^5$; para el caso de las bases de datos NSLKDD y ENSLKDD, con 41 dimensiones, éste tipo de transformación se vuelve poco práctica.

En contraste con lo dicho anteriormente, resulta que no hay necesidad *explícita* de trabajar en un espacio de dimensión mayor h, en el entrenamiento o evaluación del clasificador. Durante la evaluación de una instancia χ_n el problema de optimización realiza el cómputo únicamente del producto

punto entre $\langle \chi_n, x_n \rangle$ [54] [59]. Debido a ésto existen funciones que dado dos vectores x_n y χ_n en $\Re^p$ *implícitamente* realizan el cómputo del producto punto de x_n y χ_n en una dimensión mayor $\Re^h$ sin la necesidad de ϕ y transformar *explícitamente* x_n y χ_n a $\Re^h$. Éstas funciones son llamadas *Kernel*, $K(\chi_n, x_n)$. El Kernel K es una función que realizan el cómputo del producto punto en un espacio dimensional $\Re^h$ permaneciendo en $\Re^p$, dónde $h > p$. Es decir y expresado con formalismo, para cada $(\chi_n, x_n) \in \Re^p$, existe una función K tal que, $K(\chi_n, x_n) = \langle \phi(\chi_n), \phi(x_n) \rangle_h$, dónde $\langle \phi(\chi_n), \phi(x_n) \rangle_h$ es el producto interno de $\Re^h$, $h > p$ y cómo se mencionó anteriormente, $\phi(x_n)$ transforma x_n a $\Re^h$ es decir, $\phi : \Re^p \to \Re^h$. El hecho de que es posible obtener una clasificación *no lineal* utilizando únicamente el producto punto en una clasificación SVC, se le llama "Trick", y por ende Kernel Trick.

En consecuencia el clasificador Support Vector Machines f_{svm}, es capaz de clasificar una instancia de evaluación χ_n, mediante la ecuación (2.16)

$$f_{svm}(\chi_n) = \beta_0 + \sum_{i=1}^{n} \alpha_i K(\chi_n, x_i) \tag{2.16}$$

Dónde los parámetros β_0, y α_i se encuentran mediante los productos punto entre todos los pares $\binom{n}{2}$ de elementos $\langle x_i, x_i' \rangle$ en la base de datos de entrenamiento; el kernel K utilizado en éste trabajo de tesis es el kernel Radial Basis Function (RBF), también llamado kernel Gaussiano, es definido en la ecuación (2.17).

$$K(\chi_n, x_n) = exp\left(-\gamma \sum_{j=1}^{p} (\chi_j - x_{ij})^2\right) \tag{2.17}$$

El parámetro γ es una constante positiva la cual puede entenderse cómo aquella que determina un intervalo de decisión más focalizado o generalizado, es decir, controla el nivel de sobre entrenamiento o poco entrenamiento en el sistema.

Para ejemplificar el funcionamiento de f_{svm}, si en la ecuación (2.16) la instancia de evaluación χ_n se encuentra lejos de x_i en términos de la distancia euclidiana, entonces $\sum_{j=1}^{p} (\chi_j - x_{ij})^2$ será muy grande, y por lo tanto $K(\chi_n, x_n) = exp\left(-\gamma \sum_{j=1}^{p} (\chi_j - x_{ij})^2\right)$ será un valor muy pequeño. Ésto significa que en (2.16) la instancia x_i no tendrá ningún efecto significativo en la clasificación de $f_{svm}(\chi_n)$, ya que ésta se define por el signo de f_{svm}, si $f_{svm} \geq 0$ pertenece a una clase y si $f_{svm} < 0$ pertenece a la otra clase, de lo antedicho se desprende que sólo aquellas instancias x_i cercanas a χ_n tendrán un efecto en su clasificación.

El Algoritmo 2.2 muestra el proceso de cómputo del algoritmo Support Vector Machines con el kernel RBF, para dos o más clases.

Los valores de C, γ con los cuales se realizan los experimentos en el capí-

Algoritmo 2.2 Clasificación mediante SVM y kernel RBF

```
Xtra[]=Base de datos de entrenamiento
Xtest[]=Base de datos de evaluación
Y[]=Etiquetas correspondientes a las clases en Xtra
Yt[]=Etiquetas predecidas para las instancias en Xtest
α_i[]=paramétros de los pesos del hiperplano
f_svm = β_0
for j in length(Xtest) do
    for i in length(Xtra) do
        K=RBF(Xtest(j),Xtra(i))
        f_svm=f_svm+α(i)
    end for
    if FuncionSGN(f_svm)==1 then
        Yt(j)=1
    else
        Yt(j)=-1
    end if
    f_svm = β_0
end for
```

tulo 4 de éste trabajo de tesis, fueron encontrados a través de una búsqueda por GridSearch, precisamente $C = 10$ y $\gamma = 0.1$.

2.5.9. Decision Trees

Decision Trees (árboles de decisión, en español) es un algoritmo de la rama de técnicas de aprendizaje de máquina capaz de realizar métodos de clasificación y regresión, dónde el primero es utilizado en éste trabajo de tesis. Es un algoritmo *no paramétrico*, de ahí que no asume ninguna forma previa de f y por lo tanto ninguna distribución de la base de datos con la cual es evaluado. Es, además, el componente principal del algoritmo de aprendizaje de máquina Random Forest, descrito en la Sección 2.5.10. Decision Trees (DT) es utilizado paralelamente cómo un método del tipo filtro para reducir dimensiones en una base de datos de entrenamiento/evaluación debido a su cualidad de encontrar aquellas dimensiones que aportan mayor información, teniendo un grado de bajo *impureza*.

El algoritmo DT crea un árbol cómo el de la Figura 2.26, dónde comienza a crear de forma iterativa ramificaciones a partir de aquella separación que otorgue aquellos nodos con la mayor pureza G. Ésta medida de pureza se obtiene a través del análisis de la entropía de cada una de las dimensiones d en la base de datos de entrenamiento X. Cada uno de los nodos tanto el principal cómo los internos, se encargan de separar a las clases contenidas en X a través de una serie de preguntas. Una vez que se llega a un nodo hoja, la

clasificación de las instancias x_n contenidas en dicho nodo se realiza mediante la mayoría de elementos que pertenezcan a dicha clase, en otras palabras, en un nodo hoja pueden existir múltiples instancias x_n que pertenezcan a diferentes clases, la clase determinante de dicho nodo hoja será aquella en la que se encuentren la mayoría de instancias x_n. Cuando un nodo hoja contiene una única clase se considera un nodo *puro*, en caso contrario se considera un nodo *impuro*.

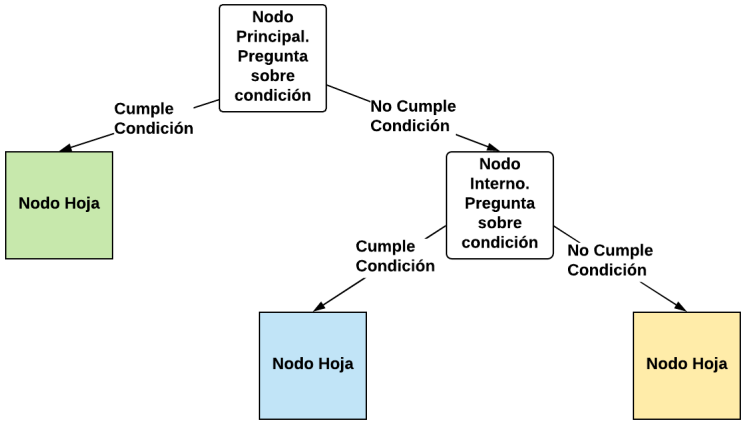


Figura 2.26: Esquema del algoritmo de ML Decision Trees

Existen diversos algoritmos para realizar la separación de un nodo principal en nodos secundarios a partir del grado de impureza G, *id3* y el algoritmo *Classification And Regression Tree* (CART), son los más representativos. En éste trabajo de tesis se utiliza el algoritmo CART debido al ser implementado en las librerías de ScikitLearn de las cuales se hace uso en los Capítulos 3 y 4. El algoritmo CART es utilizado para entrenar árboles de decisión dónde cada separación es dada por exactamente *dos nodos secundarios*. CART primero separa la base de datos X en dos subsets más pequeños X_a y X_b utilizando una *única* dimensión/característica d_i y un umbral t_i. La selección de d_i y t_i es a través de la búsqueda de aquellos pares (d_i, t_i) los cuales producen los subsets X_a y X_b con la mayor pureza. La función que el algoritmo trata de minimizar para realizar ésta separación se describe en la Ecuación (2.18)

$$DT\left(d_i, t_i\right) = \frac{m_{izq}}{m} G_{izq} + \frac{m_{der}}{m} G_{der} \tag{2.18}$$

Dónde G_{izq} es la medida de impureza del subset izquierdo, mientras que G_{der} del derecho; m_{izq} es la cantidad de instancias en el subset izquierdo y m_{der} del derecho. CART realiza la Ecuación (2.18) de forma iterativa y se detiene una vez que alcance la máxima profundidad M_p dada, o no pueda encontrar una separación que reduzca la impureza. La Ecuación (2.19)

describe la impureza de los nodos una vez realizada la separación en base a la entropía, $P_{i,k}$ es el promedio de clases k entre las instancias de entrenamiento en el nodo i-ésimo.

$$GH_i = - \sum_{\substack{k=1 \\ P_{i,k} \neq 0}}^{n} P_{i,k} \log\left(P_{i,k}\right) \tag{2.19}$$

La Figura 2.27 muestra una base de datos X con 100 instancias de las cuales, 50 pertenecen a la primer clase mostrada con un círculo mientras que las restantes 50 pertenecen a la segunda clase, definida a través de un triangulo.

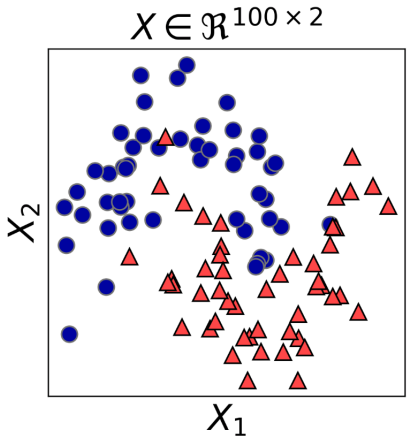
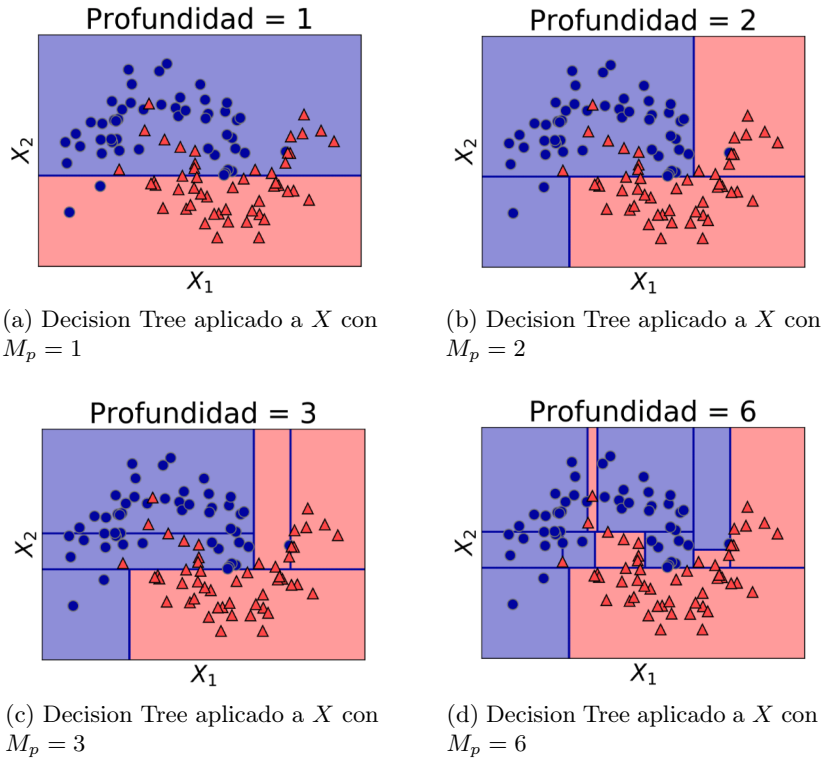


Figura 2.27: Base de datos de entrenamiento $X \in \Re^{100 \times n}$

Al aplicar el algoritmo DT a la base de datos X de la Figura 2.27, resulta la Figura 2.28 en la cual se muestra el efecto de diversos valores de M_p en la clasificación del algoritmo DT. Un valor pequeño de M_p cómo del de la Figura 2.28a y 2.28b, provoca que el algoritmo DT sea muy generalizado, es decir, puede estar poco entrenado y no proporcionar resultados adecuados en la clasificación de una base de datos de evaluación χ, en contraste la Figura 2.28d muestra el efecto de un valor de M_p alto, dónde el algoritmo DT se encuentra sobrreentrenado y únicamente proporciona una clasificación adecuada en aquellos valores que se asemejen perfectamente a la base de datos X, siendo incapaz de clasificar de manera adecuada valores desconocidos.



(a) Decision Tree aplicado a X con $M_p = 1$

(b) Decision Tree aplicado a X con $M_p = 2$

(c) Decision Tree aplicado a X con $M_p = 3$

(d) Decision Tree aplicado a X con $M_p = 6$

Figura 2.28: Efecto de la variación de la máxima profundidad M_p en el algoritmo Decision Trees

.

En el Algoritmo 2.3 se puede encontrar el procedimiento de la clasificación a través del algoritmo DT. Con altos valores de M_p el algoritmo tiende a sobreentrenarse, y su clasificación en instancias dónde no se tenga un conocimiento previo se muestra pobre. Para solventar ésta problemática el algoritmo Random Forest utiliza múltiples Decision Trees para construir un clasificador que no éste poco generalizado a través de una técnica de ensamble llamada *bootstrap*. El valor de M_p utilizado en los capítulos 3 y 4, se obtiene mediante una búsqueda de GridSearch, siendo $M_p = 10$.

2.5.10. Random Forests

De lo mostrado en la Figura 2.28 es posible observar que el algoritmo Decision Trees tiende a estar sobreentrenado a excepción de que se establezca un valor bajo de M_p, en contraste se utilizan técnicas de ensamblaje con el objeto de obtener Decision Trees diversos el uno del otro, y por ello, intervalos de decisión diferentes los unos a los otros en procura de encontrar un equilibrio entre sobreentrenamiento y poco entrenamiento. Los métodos de *ensamblaje* combinan múltiples modelos de algoritmos de aprendizaje de

Algoritmo 2.3 Clasificación mediante Decision Trees

```
1: Xtra[]=Base de datos de entrenamiento
2: Xtest[]=Base de datos de evaluación
3: Impureza=0
4: Dm=10^2. Valor alto de impureza
5: while (i<MaximaRamificacion or Impureza!=0) do
6:     d=DimensionAleatoria(Xtra)
7:     t=IntervaloAleatorio(Xtra)
8:     DT=EvaluacionGanancia(d,t)
9:     if DT<Dm then
10:        Dm=Dt
11:        CreacionDeNuevoNodo
12:    end if
13:    i++
14: end while
15: Clasificación=IntervalosDeDecisionObtenidosEnDT(Xtest)
```

máquina para crear modelos más robustos. Existen dos modelos de ensamblaje los cuales utilizan DT cómo sus algoritmos principales, Random Forests (RF) y Gradient Boosted Decision Trees. Random Forests es ocupado en éste trabajo de tesis al ser el algoritmo más utilizado en la literatura entre estos dos métodos de ensamblaje. Random Forests (Bosques Aleatorios, en español) es un algoritmo de aprendizaje de máquina *no paramétrico* dentro de los métodos de ensamblaje diseñado para solventar la problemática del sobreentrenamiento del algoritmo DT. Random Forests es en esencia un conjunto de Decision Trees, dónde cada Decision Tree es diferente de los demás. Se basa en la idea de que cada árbol puede tener un porcentaje de exactitud en la evaluación de su clasificación medianamente bueno aún cuando tienda a sobreentrenarse. Si se crean múltiples Decision Trees dónde cada uno tendrá un porcentaje aceptable de exactitud y serán sobreentrenados en diferentes maneras, se puede reducir el sobreentrenamiento general al realizar el promedio de cada Decision Tree.

De lo antedicho se desprende que RF inyecta una cierta aleatoriedad en la creación de los árboles para asegurar que cada uno de ellos es diferente. Existen dos técnicas para asegurar que cada uno de los subsets de entrenamiento de X utilizados para cada DT sean diferentes, *Sampling* y *Bootstrap Aggregating* (llamado simplemente *Bagging*) son las técnicas correspondientes para dicha tarea. El método de sampling toma instancias aleatorias de X para cada DT entrenado con un subset X_n diferente, sin embargo, ninguna de éstas instancias puede ser repetida múltiples veces en el mismo subset X_n. Al contrario, el método Bagging crea un subset X_n del mismo tamaño de X, dónde las instancias de X pueden estar repetidas múltiples veces en el

mismo subset X_n. Para ilustrar y ejemplificar, si a un set de entrenamiento $H = [a, b, c, d]$ se le es aplicado el método de Bagging, los subsets de entrenamiento podrían ser $H_1 = [b, d, d, c]$, $H_2 = [d, a, d, a]$. La Figura 2.29 ofrece una mejor representación de la clasificación RF utilizando el método de ensamblaje a través del método Bagging.

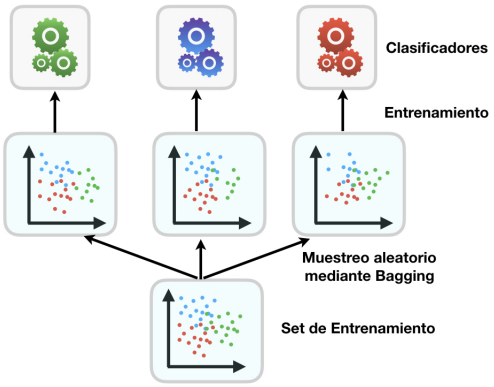


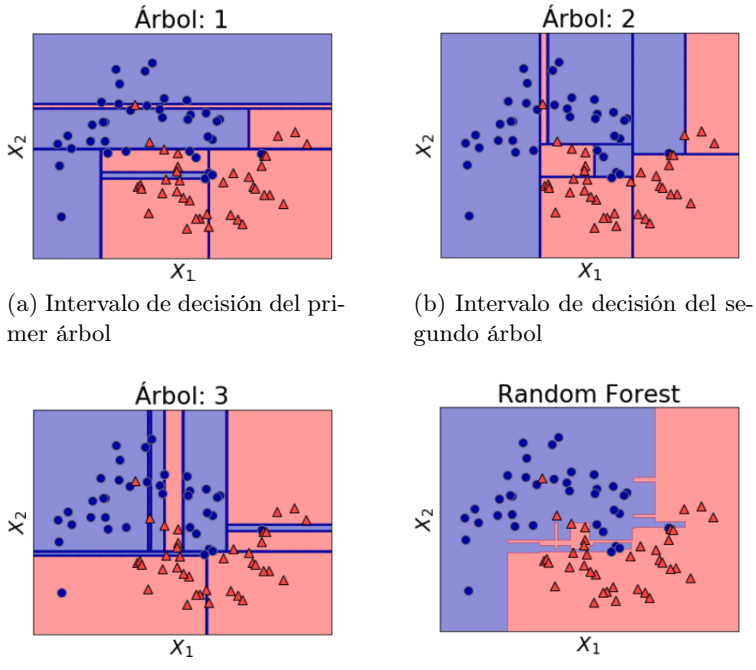
Figura 2.29: Clasificación mediante el método de ensamblaje haciendo uso del muestreo aleatorio Bootstrap Aggregating

Una vez que se han generado los subsets aleatorios X_n mediante la técnica de muestreo aleatorio Bagging, un nuevo árbol de decisión DT_n es creado con cada uno de los subsets X_n. Para la creación de los árboles de decisión DT_n pertenecientes al método RF, al contrario del método DT dónde son seleccionadas todas las dimensiones p en X cada vez que se crea un nodo, en los árboles DT_n se selecciona un subconjunto aleatorio p_r del total de dimensiones p de tamaño M_f, en otras palabras, cada nodo en un árbol puede realizar una decisión de su separación evaluando un subset de dimensiones diferente.

Para realizar la clasificación de una instancia de evaluación χ_n mediante RF, el algoritmo realiza la clasificación de χ_n para *cada árbol* en el "bosque"; a continuación realiza una técnica llamada *Soft Voting* la cual calcula la probabilidad de que χ_n pertenezca a alguna clase k y realiza su promedio. La clase k con la mayor probabilidad es seleccionada y otorgada a la instancia χ_n.

De la base de datos de entrenamiento X de la Figura 2.27 se le es aplicado el algoritmo RF con una cantidad de árboles $C_a = 5$, a partir del cual se obtiene La Figura 2.30 dónde se muestra el intervalo de decisión de los primeros tres árboles, mientras que la Figura 2.30d muestra el conjunto de los intervalos de decisión otorgados por cada árbol en Random Forest, en particular se observa un intervalo de decisión más generalizado a diferencia de las Figuras 2.30a- 2.30c, a estos efectos se obtiene un sistema no sobreen-

trenado capaz de evaluar instancias de evaluación que no se encuentren en la base de datos de entrenamiento.



(a) Intervalo de decisión del primer árbol

(b) Intervalo de decisión del segundo árbol

(c) Intervalo de decisión del tercer árbol

(d) Intervalo de decisión de Random Forest creado con cinco árboles

Figura 2.30: Intervalos de decisión creados con múltiples árboles para formar el intervalo de Random Forest utilizado para clasificación

.

El Algoritmo 2.4 muestra el proceso que lleva a cabo el algoritmo clasificador Random Forest al momento de evaluar una instancia de prueba χ_n

El algoritmo RF sufre mucho menos del problema de sobreentrenamiento que cada uno de los árboles individuales creados en la Figura 2.28. Comúnmente se utiliza una escala de decenas o centenas de árboles en RF, los cuales crean intervalos de decisión más suaves. Para el capítulo 3 y 4 se utiliza el valor de $M_p = 10$ de la sección 2.5.9, además de una cantidad de árboles $C_a = 8$. Todos éstos parámetros son obtenidos mediante una búsqueda por GridSearch.

Algoritmo 2.4 Clasificación mediante Random Forests

```
1: Xtra[]=Base de datos de entrenamiento
2: Xtest[]=Base de datos de evaluación
3: Mp= Máxima Profundidad de cada árbol
4: Ca=Cantidad de Árboles
5: for length(Subsets)<=Ca do
6:     Subset[]=MuestreoViaBagging(Xtra)
7:     for i<Mp do
8:         Dimensiones=SeleccionAleatoriaDeDimensiones(Xtra)
9:         Arbol[]=Dimensiones(Subset)
10:        ModeloArbol[]=AlgoritmoArbolesDecision(Arbol)
11:    end for
       ModeloRF[]=[ModeloRF,ModeloArbol[]]
12: end for
       Clasificación=ModeloRF(Xtest)
```

2.6. Reducción de dimensiones

De forma general los problemas que hacen uso de algoritmos de aprendizaje de máquina involucran una cantidad de dimensiones p de al menos el orden de las decenas, tal es el caso de 40 dimensiones para el caso de las bases de datos NSLKDD y ENSLKDD utilizadas en éste trabajo de tesis para la evaluación de la exactitud de los sistemas IDS. Un alto grado de dimensiones no sólo hace la parte del entrenamiento y evaluación de los clasificadores extremadamente lenta, por ejemplo, en el caso de los algoritmos KNN (Capítulo 2.5.4) y SVMs (Capítulo 2.5.8) los cuales hacen uso de algún tipo de distancia y su tiempo de cómputo está directamente relacionado con la cantidad de dimensiones, además, es mucho más complicado obtener un buen porcentaje en la evaluación de la exactitud de los clasificadores, a ésta problemática en la literatura se le llama *curse of dimensionality* (la maldición de la dimensionalidad, en español).

Debido a que la cantidad de dimensiones en el entrenamiento de los clasificadores es un problema serio, existen dos técnicas enfocadas a realizar una reducción de dicha cantidad de dimensiones p, Extracción de Características y Selección de Características.

El objetivo de la técnica Extracción de Características es transformar el conjunto de entrenamiento X el cual se considera de un espacio dimensional p alto, a un espacio de menores dimensiones p_m, a través de alguna función ϕ, dónde algunas de éstas funciones hacen uso del Kernel Trick, mismo del Capítulo 2.5.8, en otras palabras, la técnica Extracción de Características crea *nuevas características (dimensiones)* del conjunto de características original p en X; la técnica más conocida de Extracción de Características es el

Análisis por Componentes Principales (PCA).

En contraste la técnica Selección de Características se enfoca en encontrar un subconjunto p_i del conjunto p en X dónde $p_i < p$, sin alterar de forma directa las características originales p, es decir, la reducción de la selección de características se realiza evaluando la importancia de cada una de las dimensiones p con el objeto de encontrar aquellas que aporten la mayor información y prescindiendo de aquellas redundantes. Existen dos métodos principales en la técnica de Selección de Características, métodos del tipo filtro y métodos del tipo envoltura (wrapper, en inglés).

Los métodos del tipo filtro seleccionan un subset p_i del conjunto p basado en características propias del conjunto de entrenamiento X, utilizando criterios cómo la ganancia de información y de manera independiente al algoritmo clasificador utilizado. Tienen la capacidad de trabajar con bases de datos de entrenamiento grandes y con una cantidad de dimensiones p considerable. De forma general son computacionalmente menos demandantes que los métodos del tipo envoltura, en cambio, el subconjunto de dimensiones reducidas p_i encontradas suele no propiciar los mejores resultados en la evaluación de la exactitud de los clasificadores [20].

Los métodos del tipo Envoltura contrariamente a los métodos del tipo Filtro involucran en la búsqueda del conjunto de dimensiones p_i la evaluación de los resultados de exactitud de los clasificadores con dicho conjunto p_i, dicho de otra manera, los métodos de tipo Envoltura usan de manera activa los resultados de los clasificadores para determinar aquellas dimensiones que aporten la mayor información y eliminando aquellas que resulten redundantes. Al utilizar de forma activa el clasificador para determinar el mejor conjunto de dimensiones p_i, otorgan los métodos de Envoltura mejores resultados que los métodos del tipo Filtro, sin embargo, al realizar el entrenamiento y evaluación del clasificador con cada conjunto de dimensiones p_i de forma iterativa, su demanda y por ende tiempo computacional son mucho mayores.

Es preciso señalar, *ambos métodos* de reducción de dimensiones a través de reducción de características provocan una disminución en la evaluación de la exactitud de los clasificadores, de ahí que es de suma importancia determinar aquel conjunto de dimensiones p_i que merme lo menos en lo posible los resultados del clasificador y paralelamente disminuya de forma considerable los tiempos de cómputo. Los sistemas de red de detección de intrusos al clasificar de forma continua en una implementación real cada registro de conexión es imprescindible que se evalúen únicamente aquellas características (dimensiones) que aporten la mayor información de forma que se maximice la detección de intrusos minimizando los tiempos de procesado.

Lo antedicho es uno de los objetivos principales de éste trabajo de tesis, realizar un análisis de exactitud de los sistemas IDS basados en anomalías, mediante las bases de datos NSLKDD y ENSLKDD con 40 dimensiones y

un subconjunto de dimensiones reducidas p_i minimizando los tiempos de cómputo de los algoritmos clasificadores y reduciendo en lo menos posible su exactitud.

Los métodos del tipo Envoltura al trabajar de forma directa con los algoritmos clasificadores y por ende otorgar un mejor resultado en la exactitud de la evaluación de los sistemas IDS en comparación con los métodos del tipo Filtro, son utilizados en éste trabajo de tesis. Para la aplicación de éstos métodos de tipo envoltura se utilizan técnicas de búsqueda heurística, las cuales permiten reducir el espacio de búsqueda a través de una búsqueda guiada.

2.6.1. Métodos de Búsqueda Heurística

La *búsqueda* se define cómo el proceso de evaluar las distintas secuencias de acciones (espacio de estados) para encontrar aquellas que llevan del estado inicial a al estado final b, maximizando o minimizando un test objetivo a través de una función de evaluación (objetivo) f_{it} (Fitness, por su definición en la literatura). El proceso de búsqueda comienza, por tanto, definiendo el espacio de estados, a continuación se establece un estado inicial, un test objetivo y un costo del camino, el cual generalmente son iteraciones en el algoritmo de búsqueda o evaluaciones de la función f_{it} [60]. La formulación del proceso de búsqueda se ejemplifica en la Figura 2.31

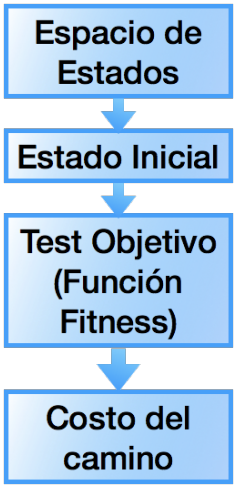


Figura 2.31: Formulación del proceso de Búsqueda

La búsqueda puede ser del tipo *no informada* cuando aquel problema que se desea resolver no ofrece ninguna información adicional que nos ayude a encontrar una solución más rápida, y del tipo de búsqueda *informada* dónde en contraste con la búsqueda no informada, el problema a solucionar

ofrece por su naturaleza información la cual resulta útil para converger a una solución de forma más rápida [61]

La búsqueda heurística, cuyo significado viene del griego *heuriskein* y significa encontrar, es una *búsqueda informada* orientada a reducir la cantidad de búsqueda requerida para encontrar una solución. Feigenbaum y Feldman definen a la heurística cómo "Una regla para engañar, simplificar o para cualquier otra clase de ardid el cual limita drásticamente la búsqueda de soluciones en grandes espacios de estados" [62].

La búsqueda heurística por tanto, se encarga de maximizar o minimizar la función objetivo f_{it} dependiendo de los requisitos del problema, a través de un conjunto de parámetros en el dominio de dicho problema los cuales afectan el valor de la función objetivo y al mismo tiempo utilizando un conjunto de restricciones las cuales se encargan de limitar los valores en el dominio que se pueden asignar. Dicho de otra manera, el objetivo de la búsqueda heurística es asignar valores del domino (aquellos que sean permitidos por las restricciones) de manera que la función objetivo f_{it} sea optimizada.

Dentro de la búsqueda heurística existen métodos de *búsqueda local* también denominados *constructivos voraces*. El objetivo de éstos algoritmos de búsqueda local (voraz) es mejorar una solución de manera iterativa, realizando pequeñas modificaciones en cada iteración sobre la última solución en el espacio de búsqueda, siempre que no se viole ninguna restricción del domino del problema. Si la modificación ofrece una mejor solución que la anterior, ésta nueva solución se conserva para iterar sobre ella, en caso contrario la nueva solución se descarta y se vuelve a iterar sobre la solución anterior. En otras palabras, en las búsquedas locales se trabaja sobre un único estado, al que se le aplica alguna modificación con el objeto de que este nuevo estado resultante se aproxime de mejor manera a la solución; a éste estado resultante del cual proviene de la modificación del estado anterior, se le llama estado vecino.

Dichos algoritmos constructivos voraces, utilizan una función heurística la cual consiste en elegir la mejor opción en cada paso para ir construyendo una solución. Son utilizados con el objeto de a partir de una solución inicial ir mejorando dicha solución con cada iteración a través de algoritmos cómo Random Mutation Hill Climbing (RMHC), Simulated Annealing (SA) e Ions Motion Algorithm (IMO) todos descritos y utilizados en éste trabajo de tesis con el fin de realizar un análisis de la exactitud de los sistemas IDS con un sistema con 40 dimensiones y otro con dimensiones reducidas.

Existen problemas representativos en la literatura para la búsqueda heurística los cuales son una generalización de situaciones las cuales se encuentran en múltiples casos, *Travelling Salesman Problem* (Problema del Viajero Vendedor, en español) se refiere al tipo de problemas de búsqueda heurística para minimizar (en éste caso distancias) dónde el espacio de búsqueda se encuentra bien definido y dónde cada nodo vecino es una especie de con-

mutación del nodo anterior; *Knapsack Problem* (Problema de la mochila, en español), es un problema de búsqueda heurística enfocado a la minimización de una función de costo (definida por el peso de la mochila) maximizando una función objetivo (definida por la cantidad de elementos en la mochila) [63]. El problema de la mochila es un tipo de problema binario, dónde un elemento de la mochila puede o no encontrarse y cuyo espacio de estados se define cómo $E_p = 2^e$ dónde e es la cantidad de elementos en la mochila. La Figura 2.32 muestra el problema de la mochila con 5 elementos, partiendo de un nodo principal y 4 posibles vecinos cercanos.

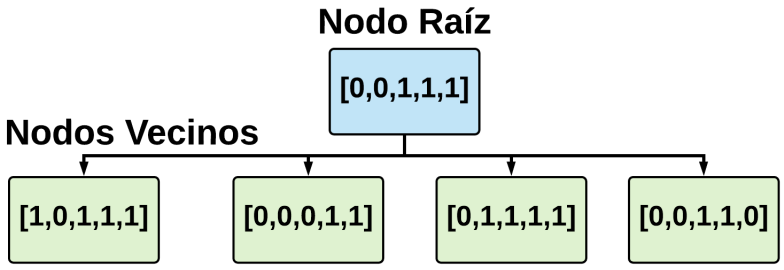


Figura 2.32: Problema de optimización: Mochila (Knapsack)

Realizar una búsqueda elemento a elemento en el espacio de estados cuando $e = 16$ se vuelve computacionalmente muy demandante ya que $E_p = 2^{16} = 65536$ para el caso de éste problema de tesis dónde $e = 40$ y por lo tanto $E_p = 2^{40} = 1,099,511,627,776$ resulta imposible realizar una búsqueda elemento a elemento y por tanto se utilizan técnicas de búsqueda heurística para reducir el espacio de búsqueda.

Dicho de otra manera, el problema de la mochila es un análogo del problema de éste trabajo de tesis, dónde la existencia de algún elemento e_i en la mochila se traslada a la existencia de una dimensión p_i en la base de datos NSLKDD y ENSLKDD y cuyo objetivo es minimizar la cantidad de dimensiones p maximizando la exactitud del algoritmo clasificador. En el capítulo 3 se describe de forma más detallada el proceso del problema de la mochila y la búsqueda heurística aplicada en éste trabajo de tesis.

2.6.2. Random Mutation Hill Climbing

Random Mutation Hill Climbing (ascenso de colinas con mutación aleatoria, en español) abreviado RMHC, es un algoritmo de búsqueda heurística de trayectoria simple (únicamente contempla un único estado por iteración), el cual se basa en el algoritmo de búsqueda heurística Hill-Climbing (ascenso de colinas, en español). Ascenso de colinas es un algoritmo el cual toma su nombre de la analogía con un montañista el cual busca llegar a la cima de

la montaña de la forma más rápida posible, y para lograrlo el montañista escoge la dirección cercana a él que brinde un mayor ascenso a partir de su posición actual [64].

A fin de encontrar la ruta más rápida posible para ascender a la montaña la estrategia del algoritmo es iterativamente expandir un nodo e inspeccionar *todos* sus nodos vecinos, si el nodo vecino ofrece una mejor posición que el nodo actual, se conserva la nueva posición del nodo generado y comienza a explorar sobre él, en caso contrario se regresa a la posición original y se prueba con otro nodo vecino. La Figura 2.33 muestra de forma gráfica el algoritmo de búsqueda de ascenso de colinas.

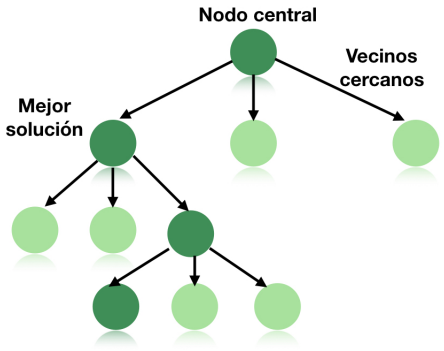


Figura 2.33: Algoritmo de búsqueda Ascenso de Colinas

Éste método de búsqueda heurística únicamente toma en cuenta los nodos vecinos y sucesores al nodo original, no mantiene ningún tipo de referencia de nodos anteriores, a diferencia del tipo de búsqueda en profundidad y amplitud. Ascenso de colinas tiene un tipo de estrategia denominada *irrevocable* porque una vez que se ha escogido un nodo sucesor a través de un nodo vecino, *no es posible* considerar algún otro nodo vecino del paso anterior y por lo tanto no es posible explorar dicho espacio de búsqueda.

Ésta situación dónde no es posible realizar un tipo de retroceso a menos que se genere un nodo raíz, es de las principales problemáticas del algoritmo ascenso de colinas debido a que la solución final es posible que se encuentre en un máximo/mínimo local del espacio de búsqueda en vez del máximo/mínimo global. Lo antedicho se puede observar en la Figura 2.34 dónde la solución final se encuentra en un máximo local del espacio de búsqueda y está limitada en encontrar el máximo global debido a las peores soluciones en sus vecinos cercanos.

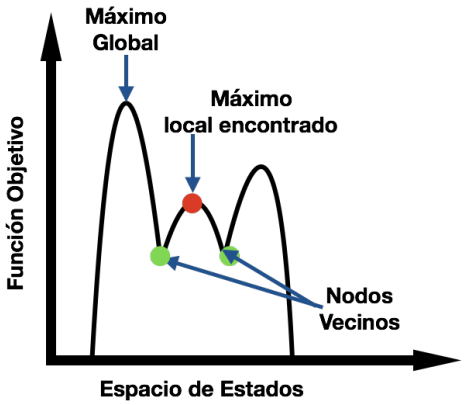


Figura 2.34: Máximo local y Máximo global en el espacio de estados mediante la función de optimización

La definición del algoritmo de búsqueda de Ascenso de Colinas tiene cómo objetivo ser una introducción para el algoritmo de búsqueda RMHC utilizado en éste trabajo de tesis.

RMHC cómo se mencionó con anterioridad, es una variante del algoritmo Ascenso de Colinas dónde a partir de un estado específico del espacio de estados se genera el siguiente estado no haciendo un análisis de cada uno de los vecinos cercanos en contraste con Ascenso de Colinas, sino su estado sucesor se genera mediante *procesos aleatorios*. En otras palabras, no se inspeccionan todos los posibles sucesores mediante los vecinos cercanos, sino únicamente el estado generado y si el estado generado supera en optimización al resultado anterior, éste nuevo estado se considera cómo el estado actual.

La Figura 2.35 muestra el proceso de búsqueda mediante el algoritmo RMHC. Es posible observar que el algoritmo Ascenso de Colinas en la Figura 2.33 hace un análisis de todos los nodos vecinos a partir del último estado generado, mientras que el algoritmo RMHC hace un único análisis del estado generado, si éste es mejor se conserva para realizar una modificación aleatoria y obtener el siguiente estado, en caso contrario se modifica aleatoriamente el estado original para encontrar otro estado sucesor.

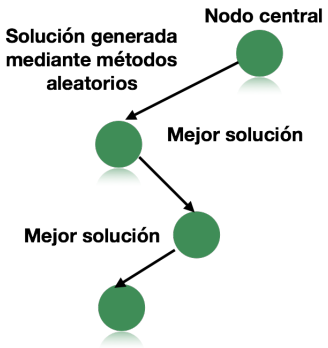


Figura 2.35: Algoritmo de búsqueda RMHC

El Algoritmo 2.5 describe el proceso de búsqueda heurística mediante RMHC.

Algoritmo 2.5 Búsqueda heurística de trayectoria simple Random-Mutation Hill-Climbing

Entrada: $P = \{P_1, P_2, \ldots, P_{40}\}$ Conjunto de Dimensiones Booleanas
Salida: $B = \{B_1, B_2, \ldots, B_{40}\}$ Mejor solución del espacio de búsqueda.

1: Se realiza la declaración de la primer exploración del espacio de búsqueda E_t con la misma cantidad de dimensiones que P
2: $E_t = \{0, 1, 0, \ldots, 0, 0, 1\}$
3: $B = E_t$
4: $i = 1$
5: **while** i<CriterioDeParada **do**
6: Evalúa el espacio de búsqueda temporal E_t mediante la
7: función de optimización
8: **if** La optimización de la función objetivo f_{it} mediante E_t ofrece un mejor resultado que B **then**
9: $B = E_t$
10: **end if**
11: Genera un nuevo elemento del espacio de búsqueda mutando
12: aleatoriamente E_t
13: **end while**

2.6.3. Simulated Annealing

Simulated Annealing (SA) ó Recocido Simulado en español, es una técnica de búsqueda heurística del tipo de trayectoria simple. Se basa en la idea del proceso de enfriamiento de los metales después de haber sido calentados para conseguir un estado idóneo denominado estado fundamental. En su definición química, es un proceso térmico para la obtención de estados de baja

energía en un elemento solido mediante un baño de calor.

El proceso de recocido simulado involucra dos partes principales. Incrementar la temperatura del baño de calor en la cual el metal solido comienza a derretirse. Después disminuir la temperatura cuidadosamente hasta que las partículas se ubiquen a si mismas en el estado fundamental del solido, en dónde éste tiene la mínima energía. El estado fundamental del solido se consigue únicamente si la máxima temperatura es lo suficientemente grande y el enfriamiento se realiza lentamente [65].

Es posible aplicar el algoritmo de recocido simulado para generar una solución a problemas de optimización asumiendo una analogía mediante el planteamiento de éstos problemas y los sistemas físicos de múltiples partículas, cumpliendo las dos equivalencias siguientes, las soluciones en el problema son equivalentes a *estados* en un sistema físico y el costo de una solución es equivalente a la energía de un estado [66].

Profundizando, para la aplicación del algoritmo de recocido simulado con fines de optimización es necesario una función sucesor la cual regrese un vecino cercano dado el nodo actual, lo cual se toma cómo analogía de la "perturbación" de las partículas en un sistema físico. Además es necesario una función objetivo a optimizar que depende del estado actual en el espacio de búsqueda, está función funge cómo la "energía" del sistema.

La búsqueda del algoritmo SA empieza con un estado aleatorio dónde siempre son aceptados movimientos en nodos vecinos los cuales mejoren el estado original, mientras que en contraste, los malos movimientos no son rechazados de forma tajante cómo en RMHC, sino son evaluados en forma de "votación" aceptando únicamente malos movimientos mediante una distribución de probabilidad de acuerdo a la temperatura del sistema. Al decrementar lentamente la temperatura, SA acepta cada vez menos malos movimientos conforme ésta disminuye, cuando la temperatura es mínima el recocido simulado aceptará unicamente aquellos movimientos que sean mejores al nodo en que se encuentra, lo que lo vuelve análogo al algoritmo RMHC, cuando SA se encuentra en el estado fundamental.

El principal problema del algoritmo RMHC y de algunos algoritmos de búsqueda heurística es la alta posibilidad de caer en máximos locales, el algoritmo SA soluciona ésta problemática aceptando de forma ocasional peores ubicaciones en el espacio de búsqueda, con la esperanza de escapar de un máximo local y llegar al máximo general. La Figura 2.36 muestra al algoritmo SA aceptando una peor posición en el espacio de búsqueda lo que le permite salir de un máximo local para llegar con suerte a un máximo global.

Figura 2.36: Algoritmo de búsqueda heurística Simulated Annealing

Cómo se ha mencionado con anterioridad SA cuenta con un sistema de *enfriamiento* el cual es parte principal del algoritmo y se encarga de determinar la probabilidad de que una mala posición sea aceptada con respecto al estado actual. El criterio de aceptación de un mal movimiento, así cómo la posibilidad de que éste sea aceptado se describe en el Algoritmo 2.6, dónde es importante hacer mención de la linea 14 la cual declara los valores K, *delta* y T_t. K se refiere la constante de Boltzmann, la cual de forma adimensional $K = 1.38064852(10)^{-23}$ y controla la probabilidad de que un mal movimiento sea aceptado; *delta* es una variable que mide la diferencia entre la posición del nodo actual con la última mejor posición, si la nueva posición es peor que la actual el valor de *delta* es igual a cero; T_t es la disminución de la temperatura en cada iteración.

De forma directa los algoritmos Ions Motion y Random Mutation Hill Climbing de búsqueda heurística utilizados en éste trabajo de tesis con el objeto de realizar una reducción de dimensiones en los sistemas de detección de intrusos mediante las bases de datos NSLKDD y ENSLKDD, *no reducen de manera explícita* la cantidad de dimensiones al ser utilizados. Ésta búsqueda para la reducción de dimensiones utiliza el método de *SA embebido en todas las otras heurísticas* gracias a su propiedad de *enfriamiento* y aceptar cada vez menos movimientos por iteración. Ésta propiedad de enfriamiento se traduce en la mutación del 90 % de dimensiones p en ENSLKDD y NSLKDD cuando se comienzan las iteraciones debido a que el sistema se encuentra en una *alta temperatura* y una disminución del 3 % de dimensiones de p en cada iteración mientras la temperatura disminuye.

Por lo antedicho, el algoritmo de búsqueda heurística SA *no se utiliza de forma independiente* en éste trabajo de tesis, sino se encuentra *embebido* en cada una de las heurísticas; el autor considera que es necesario una propia introducción del algoritmo SA ya que otros trabajos en la literatura con el objeto de reducción de dimensiones utilizando métodos heurísticos no tratan

Algoritmo 2.6 Búsqueda heurística de trayectoria simple Simulated Annealing

Entrada: $P = \{P_1, P_2, \ldots, P_{40}\}$ Conjunto de Dimensiones Booleanas
Salida: $B = \{B_1, B_2, \ldots, B_{40}\}$ Mejor solución del espacio de búsqueda.

```
Se crea un nodo de forma aleatoria en el espacio de búsqueda E_t con la misma cantidad de dimensiones que P
E_t = {0, 1, 0, ..., 0, 0, 1}
Se declara una temperatura T_t la cual disminuye en cada iteración
Se establece una temperatura T_m mínima
B = E_t
i = 0
while T_t > T_m do
    while i<IteracionesMaximas do
        Se hace el calculo de la siguiente exploración del
        espacio de búsqueda
        E_t=FuncionVecino(B)
        delta=E_t − B
        if delta>0 then
            if ValorAleatorio>=exp(-delta/K*T_t) then
                Se acepta el mal movimiento
                B = E_t
            else
                Se rechaza el mal movimiento
                B = B
            end if
        else
            Siempre se aceptan los buenos movimientos
            B = E_t
        end if
        i++
    end while
    Se disminuye la Temperatura T_t
    NuevaTemperatura(T_t)
end while
```

de reducir de forma explicita el total de dimensiones p, sino de forma *implícita* y meramente *fortuita*.

2.6.4. Ions Motion Optimization

Los algoritmos RMHC y SA (el cual se encuentra embebido en RMHC) mencionados con anterioridad son del tipo de trayectoria simple, es decir, mantienen una única solución en todo el espacio de búsqueda por cada iteración, por ende su velocidad de convergencia es mucho mayor así cómo su facilidad de implementación. En cambio, suelen caer en óptimos locales debido a una pobre exploración del espacio de búsqueda, además son poco (o nada) configurables.

Con el objeto de conseguir una mejor exploración en el espacio de búsqueda son utilizados los algoritmos del tipo poblacionales, dónde cada elemento de la población es una solución en el problema de optimización. Éstos algoritmos del tipo poblacionales entran comúnmente en la rama de la *metaheurística*, dónde éste tipo de heurística está inspirada en eventos de la naturaleza ya que se ha demostrado que los organismos vivientes son consumados solucionadores de problemas [67].

Los algoritmos del tipo metaheurísticos poblacionales son divididos por su inspiración, por ejemplo, aquellos basados en enjambres (algoritmo Particle Swarm Optimization [68]), basados en evolución (Algoritmos Genéticos) y aquellos basados en leyes físicas. El algoritmo Ions Motion Optimizacion (IMO) es un algoritmo de búsqueda heurística del tipo metaheurístico poblacional basado en leyes físicas inspirado en las propiedades de atracción de los iones en la naturaleza [69]. Está diseñado para ser de los algoritmos con los menores parámetros configurables, dónde su principal configuración es el total de elementos de la población; asimismo posee un bajo costo computacional y rápida convergencia con la facultad de escapar de óptimos locales.

El algoritmo IMO divide al total de elementos de la población en dos conjuntos de soluciones al estar basados en los dos tipos de cargas en los iones, aniones con carga negativa y cationes con carga positiva. La solución del problema de optimización representada cómo iones en la población van mejorando de manera iterativa mediante la lógica de atracción en las partículas, es decir, aquellas partículas con la misma carga se atraen y aquellas con cargas diferentes se repelen.

El principal problema de la búsqueda heurística es mantener un equilibrio entre sus dos metas primordiales, la intensificación y la diversificación del espacio de búsqueda. La diversificación se refiere al objetivo de explorar lo mayor posible el espacio de búsqueda a través de un tipo de combinación entre diferentes soluciones y cambios propios de forma brusca en las soluciones individuales. La intensificación, en caso contrario a la diversificación, las soluciones en el espacio de búsqueda son susceptibles a cambios más suaves

y fomentan conseguir una convergencia a través de las mejores soluciones obtenidas en la fase de diversificación. El algoritmo IMO posee un equilibrio entre los objetivos de diversificación e intensificación a través de dos fases principales: fase líquida, encarga de la diversificación y la fase sólida, encargada de la intensificación del espacio de búsqueda. Sin embargo pese a los resultados mostrados en [69] en el cual obtiene una mayor convergencia y una mayor calidad en los resultados de optimización de las funciones evaluadas, de acuerdo al teorema "No Free Lunch", ningún algoritmo de búsqueda es capaz de resolver de la mejor manera todos los problemas de optimización [70], lo cual impacta de forma análoga los resultados en éste trabajo de tesis al reducir las dimensiones de las bases de datos para los sistemas IDS.

Cómo se mencionó con anterioridad, la población de soluciones en el problema de optimización que hace uso del algoritmo IMO son divididas en dos grupos, uno compuesto de aniones y otro de cationes. Los iones representan las soluciones candidatas para el problema dónde las fuerzas *atracción y repulsión* son aquellas encargadas de mover ambas poblaciones de iones a través del espacio de búsqueda. Bajo ésta lógica los iones se mueven atraídos hacia aquellos *mejores iones* de la población con *cargas contrarias* y con el *mejor rendimiento*, para ejemplificar, los aniones sufrirán una fuerza de atracción hacía los mejores cationes mientras que los cationes sufrirán una fuerza de atracción hacia los mejores aniones. Las poblaciones de iones son evaluadas respecto a su rendimiento (llamado en la literatura función fitness f_{it}); en el caso de éste trabajo de tesis dicho rendimiento está definido por la exactitud de los algoritmos de aprendizaje de máquina. El mejor anión/-catión es aquel que posee el valor más alto en la exactitud de la clasificación de los registros de conexión en NSLKDD y ENSLKDD. La cantidad de movimiento que sufren las poblaciones con respecto al mejor ion se determina mediante la distancia entre ellos con el mejor ion de carga contraria. Dicho movimiento físico de *atracción/repulsión* en las partículas se ejemplifica en la Figura 2.37 [69]

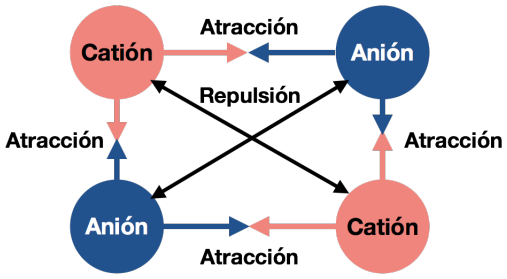


Figura 2.37: Fuerza de atracción y repulsión en los iones con diferentes cargas eléctricas

2.6.4.1. Fase líquida (Diversificación)

La fase líquida en el algoritmo IMO hace analogía al hecho físico de que las partículas en éste estado tienen mucha mayor capacidad de movimiento y por ende los iones tienen mucha mayor libertad de explorar el espacio de búsqueda guiados hacia el ion de carga eléctrica contraria con el mejor fitness F, cómo se muestra reflejado en la Figura 2.38. En otras palabras, todos los iones de la población de cationes serán aproximados al mejor ion de la población de aniones, paralelamente, todos los iones de la población de aniones serán aproximados al mejor ion de la población de cationes.

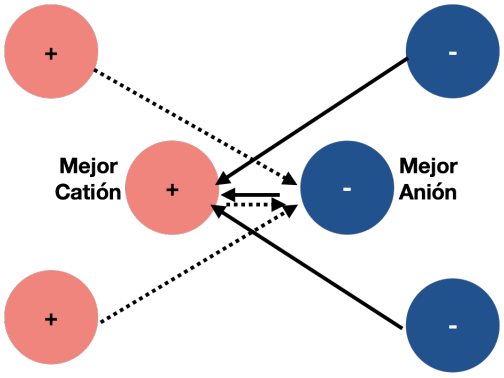


Figura 2.38: Aproximación de iones con cargas eléctricas contrarias a los mejores elementos de la población en la fase líquida del algoritmo IMO

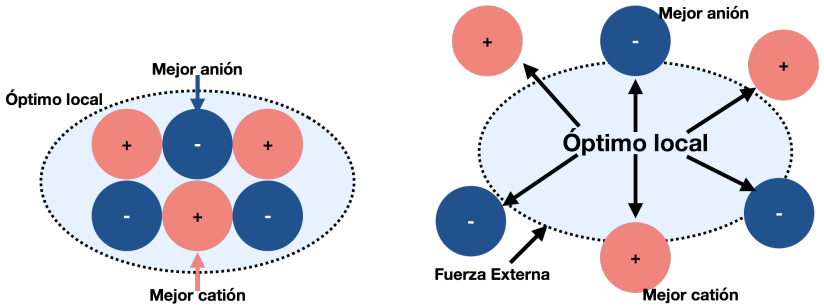
El movimiento de los iones hacia los mejores elementos de la población dependerá en última instancia de la distancia entre estos. Una distancia pequeña en un ion con respecto al mejor elemento tiene el efecto de una fuerza de atracción con mayor magnitud y por ende un brusco movimiento en la actualización de su posición. En caso contrario una distancia de mayor magnitud de un ion en la población con respecto al mejor, tendrá el efecto de una fuerza de atracción menor y por lo tanto un movimiento suave en la actualización de su posición. Los pasos para la actualización de la población de iones en el espacio de búsqueda se describen a continuación:

1. Cálculo de la distancia d_i del ion x_i con respecto al mejor elemento de la población x_{best}
2. Cálculo de la fuerza de atracción fu_i entre x_i y x_{best} mediante la distancia obtenida d_i
3. Actualización de la población mediante la fuerza fu_i

2.6.4.2. Fase sólida (Intensificación)

En la fase sólida del algoritmo IMO los iones han convergido a un punto óptimo conseguido mediante la fase líquida por lo que se asume que se ha conseguido la convergencia. Sin embargo, dicha convergencia puede ocurrir en un óptimo local debido a la dificultad de explorar todo el espacio de búsqueda. La fase solida permite a los elementos de la población salir de un óptimo local con el objetivo de llegar a un óptimo global.

La fase sólida hace analogía al hecho de que los aniones y cationes en un cristal iónico están organizados de forma que se maximiza su atracción encontrándose sumamente comprimidos generando apenas movimiento en forma vibración. Cuando una fuerza externa es aplicada a los iones en la fase sólida, se producen repulsiones entre las cargas lo que ocasiona que se agriete el sólido y los iones se separen nuevamente. La Figura 2.39 ejemplifica el agrupamiento de los iones atrapados en un óptimo local en el cristal iónico antes y después de haber sido aplicada una fuerza externa.



(a) Cationes y aniones atrapados dentro de un óptimo local.

(b) Cationes y aniones después de haber sido ejercida una fuerza externa escapando del óptimo local

Figura 2.39: Aniones y cationes agrupados en un cristal iónico antes y después de ser aplicada una fuerza externa

.

Dicha repulsión se traduce a la separación de la población de aniones y cationes alrededor de los iones con carga contraria los cuales poseen el mejor fitness, en contraste con la fase líquida la cual se encarga de guiar los iones en la población hacia los mejores.

Ésta separación puede suceder en dos casos. Una separación del elemento x_i alrededor del mejor elemento de la población x_{best} de forma suave con una distancia pequeña entre x_i y x_{best}. Y el segundo caso, una separación del elemento x_i alrededor del mejor elemento de la población x_{best} de forma violenta provocando una mayor distancia entre x_i y x_{best}. Además de ésto en cada iteración de la fase sólida existe un factor aleatorio en el cual en

caso de cumplirse tal condición *se generan nuevamente todas las soluciones* en la población de aniones y cationes, comenzando de nuevo la búsqueda de la optimización.

El proceso de búsqueda heurística IMO se muestra en el Algoritmo 2.7, dónde la linea 9 hace referencia la condición para que IMO entre a la fase sólida y comienza la intensificación del espacio de búsqueda que encontraron la población A y C en la fase líquida. La línea 10 muestra la operación *rand()*, ésta función en los lenguajes de programación arroja un valor aleatorio entre 0 y 1.

Del proceso de búsqueda heurística poblacional IMO descrito en el algoritmo 2.7 junto con el proceso de búsqueda heurística de trayectoria simple RMHC descrito en el algoritmo 2.5, es posible observar que *no existe una intención explícita* para reducir las dimensiones del problema de optimización a tratar. Es decir, IMO y RMHC *no son diseñados* de forma inherente con el objetivo de reducir dimensiones; para éste fin se implementó una analogía con el algoritmo SA en el cual cada iteración ofrece una menor cantidad de dimensiones en NSLKDD y ENSLKDD a mutar. En la sección 2.6.4 dónde es abarcado el algoritmo IMO y en la sección 2.6.2 dónde se abarca el algoritmo RMHC no se describe ninguna ecuación para implementar la reducción de dimensiones mediante SA, ésto debido a que el autor considera forma parte de la propuesta de solución en el capítulo 3. En el capítulo 3 se ahonda en la implementación de RMHC e IMO, así cómo el panorama general del objetivo de éste trabajo de tesis, realizar el análisis de la exactitud de los sistemas IDS con dimensiones completas y reducidas.

Algoritmo 2.7 Búsqueda heurística del tipo poblacional Ions Motion Optimization

Entrada: $P = \{P_1, P_2, \ldots, P_{40}\}$ Conjunto de Dimensiones Booleanas; T, cantidad de individuos en la población de aniones A y cationes C

Salida: $B = \{B_1, B_2, \ldots, B_{40}\}$ Mejor solución del espacio de búsqueda entre A y C

1: Se crean de manera aleatoria un total T de soluciones en la población de aniones A y cationes C, ambas con la forma de P
2: Es ubicado el anión a_{best} en A y el catión c_{best} en C correspondientes a aquellos individuos con la mayor exactitud del algoritmo clasificador F
3: **while** iteraciones<Condición de Parada **do**
4: Se calcula la distancia da_i de a_i con c_{best}
5: Se calcula la fuerza de atracción fa_i en a_i con c_{best} mediante da_i
6: Se actualiza la posición de a_i con respecto a c_{best} debido a fa_i
7: Se calcula la distancia dc_i de c_i con a_{best}
8: Se calcula la fuerza de atracción fc_i en c_i con a_{best} mediante dc_i
9: Se actualiza la posición de c_i con respecto a a_{best} debido a fc_i
10: **if** Al menos el 70 % de la población en A y C tienen un rendmiento mayor o igual a $.80 * (Ion_{best})$ **then**
11: **if** rand()>0.5 **then**
12: Hay una separación de forma drástica entre a_i y c_{best}
13: **else**
14: Separación mesurada entre a_i y c_{best}
15: **end if**
16: **if** rand()>0.5 **then**
17: Hay una separación de forma drástica entre c_i y a_{best}
18: **else**
19: Separación mesurada entre c_i y a_{best}
20: **end if**
21: **if** rand()<0.0005 **then**
22: Reinicia la población A y C de manera aleatoria
23: **end if**
24: **end if**
25: Se almacena la mejor solución B entre a_{best} y c_{best}
26: **end while**

Capítulo 3

Propuesta de Solución

It's tough to make predictions,
especially about the future

Yogi Berra

Resumen: El presente capítulo describe la propuesta de solución de este trabajo de Tesis al problema de la carencia de un análisis exhaustivo y replicable de los sistemas de red de detección de intrusos basados en anomalías utilizando algoritmos de aprendizaje de máquina para su evaluación mediante un sistema de dimensiones completas (utilizando las 40 dimensiones de NSLKDD y ENSLKDD) y dos sistemas de dimensiones reducidas obtenidas mediante búsqueda heurística. Paralelamente se detalla la aplicación de los dos sistemas IDS de dimensiones reducidas al problema del alto costo computacional y lento entrenamiento de los algoritmos de aprendizaje de máquina con múltiples dimensiones. Primero se muestran las mayores problemáticas a las que se enfrentan los estudios del estado del arte de los sistemas IDS, a las cuales, se propone solución. Se presenta además el panorama general de éste trabajo de tesis en forma de diagrama de flujo junto con la solución desarrollada. Después se muestra el proceso de búsqueda heurística modificado con el objeto de reducir explícitamente las dimensiones en el sistema IDS utilizando el algoritmo RMHC junto con SA. Por último se muestra de forma detallada el proceso de búsqueda heurística IMO modificado de manera original aplicado al problema binario que representan los sistemas IDS, con el objetivo explícito de reducir dimensiones.

3.1. Problemáticas en el estado del arte de los sistemas IDS y su propuesta de solución

Existen tres inconvenientes principales en el problema del análisis de la exactitud de los sistemas IDS en su estado del arte. El primer inconveniente es el uso de la obsoleta base de datos KDD99 utilizada para el entrenamiento y evaluación de los sistemas IDS en artículos recientes [69, 71, 72, 73] pese a que J. McHugh en [8] determina que los resultados de la evaluación de detección de intrusiones en una red utilizando dicha base de datos no son confiables al mantener un alto número de registros de conexión repetidos lo que provoca que los resultado sean sesgados hacia algoritmos de aprendizaje de máquina que posean la cualidad de detectar múltiples instancias repetidas. Para solucionar ésta problemática en el año 2009, Tavallaee [9] propone las bases de datos *KDDTrain+* y *KDDTest+* utilizadas y nombradas en éste trabajo de tesis *NSLKDD* y *ENSLKDD*, respectivamente. Con ellas se da solución al problema de instancias redundantes en KDD99, contando con un total de instancias lo suficientemente pequeño para realizar pruebas con las bases de datos completa, a diferencia de KDD99 dónde algunos investigadores toman únicamente fracciones de ésta parea realizar experimentos.

La segunda problemática es la falta de parámetros comunes para la evaluación de la exactitud de los algoritmos de aprendizaje de máquina para su evaluación en los sistemas IDS. Cómo se ha hecho mención, los casi 5 millones de instancias en KDD99 ha provocado que investigadores en [15, 22] utilicen porciones aleatorias de la base de datos para el entrenamiento de los clasificadores, siendo imposible replicar resultados ya que no se menciona la semilla para su segmentación aleatoria ni mucho menos se ofrece la base de datos utilizada en sus experimentos. Al mismo tiempo en algunos casos muchos más graves, la base de datos de entrenamiento KDD99 es utilizada cómo *entrenamiento y evaluación* de los algoritmos clasificadores, de ahí que los resultados obtenidos alcancen valores mayores del 98 % en el análisis de su exactitud de los sistemas IDS al no considerar la base de datos de evaluación la cual contiene ataques nuevos los cuales *no se encuentran* en la base de datos de entrenamiento, detalle de ésto es mostrado en la Tabla 2.2 de la página 25, es decir, al realizar un sistema IDS *conociendo con anterioridad* todos los ataques posibles en los registros de conexión, se pierde el objetivo principal de los sistemas IDS, el cual es detectar anomalías, y por lo tanto, ataques *desconocidos*.

En aquellos trabajos del estado del arte dónde no se hace uso de la base de datos KDD99, pero sí de NSLKDD y ENSLKDD, no se segmenta de ninguna manera las bases de datos de entrenamiento y evaluación, existe la problemática de no ofrecer ningún parámetro de configuración de los algoritmos de aprendizaje de máquina y/o la búsqueda heurística, cómo es el caso del método para la evaluación de la distancia en KNN o el total de árboles

en el algoritmo RF, de lo antedicho se desprende que sea imposible replicar resultados al no existir detalles de su implementación.

La tercer problemática es el alto número de dimensiones (40) en NSLKDD y ENSLKDD, puesto que el tiempo de cómputo al entrenar y evaluar los algoritmos de aprendizaje de máquina para los sistemas IDS depende directamente de dicho total de dimensiones. Siendo que el objeto final de los sistemas IDS es detectar una intrusión en un sistema en tiempo real, dicha velocidad de entrenamiento y evaluación en los algoritmos de aprendizaje de máquina se convierte en una meta crítica.

Éste trabajo de tesis presenta una solución a las tres problemáticas principales en el análisis de la exactitud de los sistemas IDS. Utiliza las bases de datos adecuadas NSLKDD y ENSLKDD, para el entrenamiento y evaluación de los algoritmos de aprendizaje de máquina, además de no segmentar ninguna de ellas en pequeñas porciones ni utilizar únicamente la base de datos de entrenamiento NSLKDD con el objeto de entrenar y al mismo tiempo evaluar los clasificadores. Los parámetros mostrados en éste trabajo son fácilmente replicables, se detallan las configuraciones de normalización, algoritmos de aprendizaje de máquina y de búsqueda heurística. Por último, con el objeto de realizar un análisis de la exactitud de los sistemas IDS con dimensiones completas y reducidas, se aplica la técnica de reducción de dimensiones por medio del método de la envoltura a través de la búsqueda heurística. Ésta reducción de dimensiones resulta clave en el análisis de exactitud de los sistemas IDS para evaluar la existencia de una combinación de registros de conexión específicos que otorguen la máxima exactitud posible minimizando los tiempos de entrenamiento y evaluación de los clasificadores. La reducción de dimensiones en éste trabajo de tesis se realiza a partir de los algoritmos de búsqueda heurística RMHC e IMO, dentro de los cuales se encuentra embebido el algoritmo SA. El procedimiento a detalle de la implementación para RMHC e IMO se encuentra en la sección 3.3 y 3.4 respectivamente; además, en la sección 2.6.2 y 2.6.4 se ahonda en sus orígenes junto con su algoritmo general.

3.2. Panorama general del proyecto

La primer parte de éste trabajo de tesis consiste en evaluar de manera individual el análisis de la exactitud de los algoritmos clasificadores de aprendizaje de máquina, KNN (Sección 2.5.4), SVM's (Sección 2.5.8), DT (Sección 2.5.9) y RF (Sección 2.5.10), aplicados en las bases de datos de los sistemas IDS, NSLKDD y ENSLKDD, con sus 40 dimensiones originales, es decir sin dimensiones reducidas.

La segunda parte consiste en realizar un análisis de la exactitud de los algoritmos clasificadores de aprendizaje de máquina en los sistemas IDS, aplicados a dos sistemas con dimensiones reducidas obtenidas mediante la

búsqueda heurística utilizando los algoritmos Random Mutation Hill Climbing e Ions Motion Optimization. En otras palabras, se pretende determinar aquellos conjuntos de dimensiones reducidas que otorguen los valores más altos en la exactitud de los sistemas IDS pese a no contar con el conjunto de dimensiones completo, con el objetivo de disminuir lo menos posible el valor de su exactitud reduciendo de forma significativa el tiempo de entrenamiento y evaluación de los sistemas IDS al poseer menores dimensiones que un sistema completo.

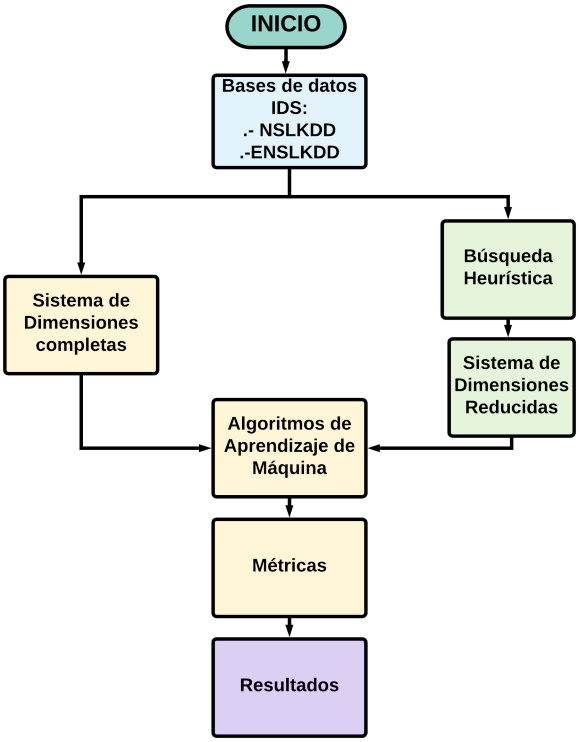


Figura 3.1: Diagrama de flujo del desarrollo realizado en éste trabajo de tesis

Dicha evaluación de la exactitud de los algoritmos de aprendizaje de máquina en los sistemas IDS tanto en el sistema con 40 dimensiones cómo en el sistema de dimensiones reducidas obtenidas mediante búsqueda heurística, se realiza a través de la clasificación en 2 clases (también llamada clasificación binaria), dónde un registro de conexión en NSLKDD puede ser considerado del tipo de conexión normal (también llamado no ataque) y cuando se considera que un registro de conexión en NSLKDD es algún tipo de ataque, sin embargo son *agrupados* los 4 tipos de ataques: DoS, U2R, R2L y Probe, en una *única* clase. A la evaluación de la exactitud de los algoritmos se añade la clasificación por 5 clases, dónde un registro de conexión en NSLKDD

puede ser considerado una conexión del tipo normal o alguna de los 4 tipos de ataques descritos con antelación. Es decir, el objetivo de los algoritmos de aprendizaje de máquina no se reduce al único hecho de detectar si ha existido algún tipo de intrusión, sino ha determinar qué tipo de intrusión ha sucedido dependiendo de las 4 clases de ataques. El análisis de la exactitud de los algoritmos de aprendizaje de máquina en los sistemas IDS con 2 y 5 clases tiene el objeto de realizar una comparativa de la exactitud del algoritmo clasificador cuando la cantidad de clases a detectar aumenta.

Las métricas utilizadas para dicho análisis de exactitud en éste trabajo son matrices de confusión [57], score del clasificador (porcentaje de instancias correctamente clasificadas) y la curva ROC únicamente para el caso de la clasificación binaria [54].

La Figura 3.1 muestra cómo panorama general el diagrama de flujo a seguir en éste trabajo de tesis. La parte izquierda del diagrama de flujo hace referencia al análisis de la exactitud de los clasificadores en un sistema con dimensiones completas, por tanto, sin ninguna modificación previa a NSLKDD y ENSLKDD. La parte derecha del diagrama muestra el proceso de reducción de dimensiones en NSLKDD a través de la búsqueda heurística; una vez realizado dicho proceso los resultados son evaluados mediante los algoritmos de aprendizaje de máquina. La Figura 3.1 describe el mismo proceso para el caso de la clasificación en 2 y 5 clases.

3.3. Random Mutation Hill Climbing y Simulated Annealing

Los sistemas IDS trabajan mediante algoritmos de aprendizaje de máquina los cuales son entrenados mediante una base de datos NSLKDD la cual contiene registros de conexión pertenecientes a carácteristicas del tráfico de la red de datos. Una vez que el modelo de *aprendizaje* ha sido creado mediante el algoritmo de aprendizaje de máquina, es evaluado frente a una base de datos de evaluación ENSLKDD la cual contiene registros de conexión *desconocidos*, en otras palabras, es evaluado el modelo de aprendizaje creado contra registros de conexión que no se tenga conocimiento previo.

Aquellas dimensiones a considerar o ignorar en el entrenamiento y evaluación de los sistemas IDS, las cuales pertenecen a características particulares de los registros de conexión del tráfico de red en NSLKDD y ENSLKDD son almacenadas en vectores binarios, denominados D_n para el caso de NSLKDD y D_e para el caso de ENSLKDD. Es decir, $(D_n, D_e) \in \{0, 1\}$ dónde el valor 0 en alguna dimensión en particular, significa no evaluar dicha característica del tráfico de red en el entrenamiento de los clasificadores, mientras que en caso contrario, el valor 1 en alguna dimensión en particular, significa considerar aquella característica del tráfico de red en el entrenamiento de los clasificadores. La Ecuación 3.1 muestra cómo ejemplo las primeras 10

dimensiones en D_n que serían utilizadas en el entrenamiento del algoritmo clasificador, dicho de otra manera, las dimensiones $2, 5, 7, 10$ representadas con el número 1 en D_n, son evaluadas en el entrenamiento del algoritmo de aprendizaje de máquina. Las dimensiones $2, 5, 7, 10$ corresponden a las características de conexión del tráfico de red: Protocol Type, Src Bytes, Land, y Hot. Éstas características de conexión del tráfico de red, cómo todas las demás características en NSLKDD y ENSLKDD, se describen en la tabla 2.3 de la página 26.

$$D_n = [0, 1, 0, 0, 1, 0, 1, 0, 0, 1] \tag{3.1}$$

RMHC es un algoritmo del tipo de *trayectoria simple*, lo que quiere decir que en cada iteración mantiene una *única solución* del espacio de búsqueda, por lo tanto cada vez que el algoritmo es evaluado utilizando cómo función *fitness* f_{it} al algoritmo clasificador RF, existe un único vector D_n y D_e. Al ser aplicada la función f_{it} a D_e se obtiene su rendimiento R_e, dicho rendimiento es descrito en la Ecuación 3.2, dónde $S(D_e)$ es el score (relación de instancias correctamente clasificadas con el total de instancias) del conjunto de dimensiones temporales utilizadas D_e a través del algoritmo RF y P_{error} es su probabilidad de error.

$$R_e = 1 - f_{it} = 1 - S(D_e) = P_{error}(D_e) \tag{3.2}$$

En cada iteración RMHC muta de manera aleatoria los vectores binarios D_n y D_e, generando un nuevo vector temporal binario Td_n y Td_e, además por tanto un rendimiento temporal de Td_e, Tr_e. Si $Tr_e < R_e$ es decir, si la exactitud del clasificador RF es mejor con el nuevo conjunto de dimensiones, al vector D_e se le asigna éste nuevo conjunto temporal Td_e, en otras palabras $D_e = Td_e$.

De lo antedicho se desprende que no existe una intención explícita de reducir las dimensiones utilizando la búsqueda heurística RMHC, por lo tanto, se propone utilizar el concepto principal del método Simulated Annealing, dónde conforme cada iteración, se reduce la posibilidad de aceptar un mal movimiento, ésto se traduce en que cada iteración de la búsqueda heurística se reduce la cantidad de dimensiones a mutar [13]. El número de características a mutar N en forma aleatoria en cada iteración está dado por la Ecuación 3.3, dónde D_{max} es el máximo número de dimensiones posibles a mutar, $I(i)$ es descrita en la Ecuación 3.4 y es una función que depende de la iteración i actual, así cómo de la máxima iteración I_{max}.

$$N = D_{max} * min\left[I(i), P_{error}(D_e)\right] \tag{3.3}$$

$$I(i) = 1 - \frac{i}{i_{max}} \tag{3.4}$$

Al utilizar un algoritmo del tipo envoltura la función R_e tiene que encontrarse en función del error de clasificación $P_{error}(D_e)$ y del número de

características restantes, de lo contrario no hay un incentivo explícito de reducir la cantidad de dimensiones [13]. Por ésta razón la función R_e se reexpresa cómo $F(D_e)$ en la Ecuación 3.5, dónde $\alpha \in \{0, 1\}$ es un factor de peso entre la reducción de dimensiones y la tasa de error, $|D_e|$ es la cardinalidad del conjunto D_e y T es el total de dimensiones originales.

$$F(D_e) = \alpha P_{error}(D_e) + (1 - \alpha)\frac{|D_e|}{T} \tag{3.5}$$

La función $F(D_e)$ en la Ecuación 3.5 depende por tanto del error de clasificación con un peso α y la fracción de características utilizadas en esa iteración. El objetivo de RMHC es llevar los dos términos de la Ecuación 3.5 a cero.

Dicho parámetro α se establece basado en cuan agresivamente RMHC reduce el número de dimensiones. Un valor grande de α fuerza a la solución final a estar basada en el error de clasificación de RF; en contraste, un valor de α pequeño alienta a la solución final a usar menos dimensiones tolerando un mayor valor de probabilidad de error de clasificación de RF.

El Algoritmo 3.1 describe el proceso de búsqueda heurística mediante RMHC y SA, utilizado en éste trabajo de tesis.

Algoritmo 3.1 Búsqueda heurística de trayectoria simple Random-Mutation Hill-Climbing con Simulated Annealing

Entrada: $P = \{P_1, P_2, \ldots, P_{40}\}$ Conjunto de Dimensiones Booleanas, D_{max}, α

Salida: $Dr = \{B_1, B_2, \ldots, B_{40}\}$ Mejor solución del sistema con dimensiones reducidas

1: Se realiza la declaración de la primer exploración del espacio de búsqueda E_t con la misma cantidad de dimensiones que P
2: $E_t = \{0, 1, 0, \ldots, 0, 0, 1\}$
3: Se declara un vector $B = E_t$, el cual será el mejor punto del espacio de búsqueda encontrado
4: $i = 1$
5: **while** i<Iteraciones Máximas **do**
6: Calcula el máximo número de dimensiones a mutar N
7: Crea un nuevo vector temporal E_t a partir de B con la mutación de N dimensiones
8: Evalúa el espacio de búsqueda temporal E_t mediante la función de optimización F y el parámetro α
9: **if** La optimización de la función objetivo $F(E_t)$ ofrece un mejor resultado que B **then**
10: $B = E_t$
11: **end if**
12: **end while**

3.4. Ions Motion Optimization y Simulated Annealing

En contraste con RMHC, Ions Motion Optimization (IMO) es un algoritmo de búsqueda heurística del tipo poblacional, es decir, en cada iteración mantendrá un *conjunto de individuos* pertenecientes a dicha población. En RMHC aquellas dimensiones a ignorar o considerar para evaluación del clasificador se denotaban por $D_e \in \{0,1\}$, dónde D_e es un único vector; en IMO los elementos de la población son representados cómo matrices E, tal que $E \in \{0,1\}^{n \times p}$ dónde n es el total de elementos en la población y p es el total de dimensiones. La Ecuación 3.6 muestra cómo ejemplo las primeras diez dimensiones en ENSLKDD mediante E en una población de $n = 6$, dónde cada individuo de la población *posee características diferentes* para evaluación del clasificador RF.

$$E = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.6}$$

IMO cómo se hizo mención en el Capítulo 2.6.4 imita el movimiento de los iones en la naturaleza, los aniones con carga negativa se aproximan a los cationes con carga positiva e iones de la misma carga se repelen. Por lo tanto existen dos poblaciones de elementos E, una población de aniones E_a y una población de cationes E_c. Paralelamente existe un vector $FE_a(x)$ y $FE_c(x)$ correspondiente a la evaluación del rendimiento F del individuo (ion) x.

IMO comienza la búsqueda heurística a través de la fase líquida encargada de la diversificación, dicha fase cuenta con tres evaluaciones, la distancia del elemento x de la población con respecto al mejor ion de la población con carga contraria b_{est}, la fuerza de atracción que depende de la distancia de los iones y por último la actualización de la posición de x que depende de la fuerza de atracción.

La distancia $d_{ist} \in \{0,1\}$ es calculada mediante el índice de Jaccard [74], dicho índice de Jaccard muestra la similitud de dos vectores *binarios*. Cuando ambos vectores son iguales, el índice de Jaccard resulta 0 otorgando la mínima distancia, en contraste, cuando ambos vectores son completamente diferentes el índice Jaccard resulta 1 otorgando la máxima distancia. El índice de Jaccard se describe en la Ecuación 3.7, dónde NTF resulta el número de dimensiones en las cuales el primer valor es *verdadero* (1) y el segundo es *falso* (0), NFT es el número de dimensiones en las cuales ambos valores son falsos y NTT es el número de dimensiones en las cuales ambos valores son verdaderos.

$$J_{accard} = d_{ist}(x, b_{est}) = \frac{NTF + NFT}{NTF + NFT + NTT} \tag{3.7}$$

La fuerza f_u de atracción entre un elemento x de la población de iones con el mejor ion de carga contraria b_{est} depende del índice de Jaccard, es decir, $f_u(d_{ist}(x, b_{est}))$. La fuerza f_u es calculada mediante una modificación de la función sigmoid $sgm(f_u) \in \{0.5, 1\}$, descrita en la Ecuación 3.8.

$$f_u = sgm(d_{ist}(x, b_{est})) = \frac{1}{1 + e^{\frac{-0.1}{d_{ist}(x, b_{est})}}} \tag{3.8}$$

La función f_u tiene dos objetivos, ser un criterio para entrar a la fase sólida del algoritmo IMO y ser el peso para decidir la cantidad de dimensiones a mutar por iteración. La Figura 3.2 muestra la gráfica de la función $sgm(d_{ist})$ la cual define la fuerza de atracción entre iones. Es posible observar que para valores mayores a $d_{ist} = 0.2$ no hay una variación drástica de la fuerza, en contraste con aquellos valores $d_{ist} < 0.2$. De lo antedicho se desprende que cuando al menos el 70 % de la población E_a o E_c, posea una $d_{ist}(x, b_{est}) < 0.1$, es decir, cuando los individuos de la población sean *muy similares* al mejor ion b_{est} y por ende se produzca una fuerza $f_u > 0.7$ en la mayoría de individuos en E_a o E_c, se entre en la fase sólida.

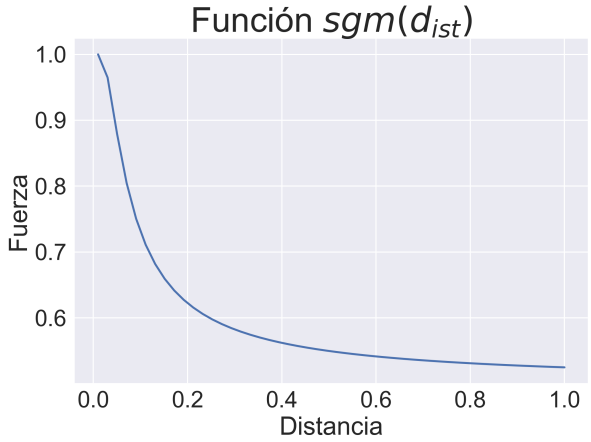


Figura 3.2: Función fuerza entre iones f_u

La actualización de la posición en la fase líquida de los elementos x en E_a y E_c, es decir, la cantidad de dimensiones a mutar, se describe en la Ecuación 3.9, dónde N se define previamente en la Ecuación 3.3. Por lo tanto, el número de dimensiones a mutar va a estar definido por la fuerza y por el número de dimensiones posibles a mutar por cada iteración.

$$P_{os}(f_u) = f_u * N \tag{3.9}$$

El criterio para entrar en la fase sólida cómo se mencionó con anterioridad, es que la fuerza de al menos el 70 % de los elementos de E_a y E_c sea mayor a 0.7, una vez que éste criterio se cumple pueden suceder tres situaciones cómo dice la Sección 2.6.4.2 y el Algoritmo 2.7, una separación drástica entre los iones, una separación mesurada o el reinicio y la generación aleatoria de las poblaciones E_a y E_c. Si un número $rand()$ generado aleatoriamente resulta mayor a 0.5 se presenta una separación drástica entre x y el mejor elemento de E_a o E_c, según la carga de x. Ésta separación drástica se traduce en la Ecuación 3.10, dónde la dispersión D_{isp} resulta un escalar correspondiente al número de dimensiones *iguales* a mutar y $|x|$ es la cardinalidad del ion x.

$$D_{isp} = N * |x| \tag{3.10}$$

Para ejemplificar, un individuo de E_a denominado e_{a_i} y un elemento de E_c denominado e_{c_i}, dónde el mejor individuo de E_a es $E_{a_{best}}$ y el mejor individuo de E_c es $E_{c_{best}}$. Al encontrarse la mayoría de elementos de E_a, e_{a_i} con una distancia muy pequeña con respecto a $E_{c_{best}}$, provocará una dispersión D_{isp} de e_{a_i} alrededor de $E_{c_{best}}$. Dicha dispersión es la cantidad de dimensiones *iguales* a mutar entre e_{a_i} y $E_{c_{best}}$ dónde D_{isp} siempre es redondeada hacia abajo. Al mutar únicamente los elementos iguales, se asegura una dispersión de e_{a_i} *alrededor* de $E_{c_{best}}$. El mismo procedimiento se realiza para los iones e_{c_i} con respecto a $E_{a_{best}}$.

En el caso dónde el número aleatorio $rand()$ resulta menor a 0.5 sucede una separación mesurada de los iones x alrededor de los mejores elementos de E_a y E_c. Ésto se traduce a una dispersión D_{isp} descrita en la Ecuación 3.11, aquí se introduce el término $\phi \in \{0.5, 1\}$, el cual es un número que se obtiene aleatoriamente; a través de ϕ es posible realizar una dispersión D_{isp} más discreta en contraste con la Ecuación 3.10, debido a que se muta una cantidad más pequeña de dimensiones similares entre e_{a_i} y $E_{c_{best}}$, así cómo e_{c_i} y $E_{a_{best}}$

$$D_{ispersion} = \phi * N * |x| \tag{3.11}$$

El Algoritmo 3.2 describe el proceso de búsqueda heurística poblacional de IMO y SA, utilizado en éste trabajo de tesis. El rendimiento F en todos los individuos de la población es obtenido a través de la Ecuación (3.5), dónde el parámetro D_e resulta ser e_{a_i} y e_{c_i} dependiendo el caso.

Algoritmo 3.2 Búsqueda heurística del tipo poblacional Ions Motion Optimization con Simulated Annealing

Entrada: $P = \{P_1, P_2, \ldots, P_{40}\}$ Conjunto de Dimensiones Booleanas; T_{ot}, cantidad de individuos en la población de aniones A y cationes C, α, D_{max}

Salida: $B = \{B_1, B_2, \ldots, B_{40}\}$ Mejor solución del espacio de búsqueda entre A y C

1: Se crean de manera aleatoria un total T_{ot} de soluciones en la población de aniones A y cationes C, ambas con la forma de P
2: Es ubicado el anión a_{best} en A y el catión c_{best} en C correspondientes a aquellos individuos el mejor rendimiento a través de la función F
3: **while** iteraciones<Condición de Parada **do**
4: Se calcula el máximo número de dimensiones a mutar N
5: Se calcula la distancia de Jaccard da_i de a_i con c_{best}
6: Se calcula la fuerza de atracción fa_i en a_i con c_{best} mediante da_i
7: Se actualiza la posición de a_i con respecto a c_{best} debido a fa_i y sujeto a N
8: Se calcula la distancia de Jaccard, la fuerza de atracción y se actualiza la posición para la población de cationes C, c_i con respecto a a_{best} debido a fc_i y sujeto a N
9: **if** Al menos el 70 % de la población en A y C tienen un fuerza mayor o igual a 0.70 **then**
10: **if** rand()>0.5 **then**
11: Hay una separación de forma drástica entre a_i y c_{best} sujeto a N
12: **else**
13: Separación mesurada entre a_i y c_{best} sujeto a N y α
14: **end if**
15: **if** rand()>0.5 **then**
16: Hay una separación de forma drástica entre c_i y a_{best} sujeto a N
17: **else**
18: Separación mesurada entre c_i y a_{best} sujeto a N y α
19: **end if**
20: **if** rand()<0.0005 **then**
21: Reinicia la población A y C de manera aleatoria
22: **end if**
23: **end if**
24: Se almacena la mejor solución B entre a_{best} y c_{best}
25: **end while**

Capítulo 4

Experimentos y Resultados

El mundo es un caleidoscopio. La lógica la pone el hombre. El supremo arte es el del azar

Miguel de Unamuno

Resumen: En el presente capítulo se muestran los experimentos realizados y los resultados obtenidos. Primero se describe el entorno en el que se desarrollan los algoritmos de aprendizaje de máquina, se mencionan las librerías utilizadas y el código en lenguaje Python implementado para la búsqueda heurística. A continuación se evalúan los algoritmos KNN, SVM's, DT y RF, en el caso de 2 y 5 clases, para determinar el clasificador que muestre el mejor equilibrio entre porcentaje de aciertos y tiempo de entrenamiento/evaluación. Después se implementa la reducción de dimensiones en NSLKDD y ENSLKDD utilizando los dos algoritmos de búsqueda heurística RMHC e IMO, con los cuales se obtiene un conjunto menor a las 40 dimensiones originales. Por último se realiza una comparativa en términos de porcentaje de exactitud y tiempo de entrenamiento/evaluación del clasificador para un sistema con 40 dimensiones y dimensiones reducidas.

4.1. Entorno de Evaluación

Todos los experimentos fueron realizados en una computadora con la configuración que muestra la Figura 4.1. El lenguaje de programación en el cual son aplicados los algoritmos de ML es Python mediante su versión *3.6.1*, utilizando además la API de Scikit-Learn [51] en su versión *0.19.1*. Los algoritmos clasificadores son implementados cómo sigue, K-Nearest Neighbors es implementado mediante la librería *sklearn.neighbors.KNeighborsClassifier* [75], Support Vector Machines denominado en Scikit-Learn *SVC* es implementado

mediante la librería *sklearn.svm.SVC* [76], Decision Trees es implementado mediante la librería *sklearn.tree.DecisionTreeClassifier* [77] y Random Forest el único método de ensamblaje es implementado mediante la librería *sklearn.ensemble.RandomForestClassifier* [78].

Figura 4.1: Características de la CPU utilizada en los experimentos

Los algoritmos de búsqueda heurística implementados en Python, RMHC e IMO son descritos en el apéndice B, mientras que su código se puede encontrar para fácil descarga en [79]. Los parámetros de configuración para los algoritmos de aprendizaje de máquina utilizando la API de ScikitLearn aplicados en éste trabajo de tesis fueron obtenidos mediante una búsqueda GridSearch, dónde la letra en *itálica* indica la configuración propia de cada una de las clases y se describen cómo sigue:

- K-Nearest Neighbors:
  - Cantidad de Vecinos, *n neighbors=10*
  - Peso de los vecinos, *weights="distance"*
  - Cálculo de la distancia, *metric="minkowski"*
- Support Vector Machines:
  - Tipo de Kernel, *kernel="rbf"*
  - Parámetro regulador de las trasgresiones al hiperplano, *C=10*
  - Parámetro regulador del intervalo de decisión, *gamma=0.1*

- Decision Trees:
  - Criterio de separación del árbol, *criterion="entropy"*
  - Máxima cantidad de dimensiones a considerar por separación, *max features="sqrt"*
  - Criterio de separación para las máximas dimensiones consideradas, *splitter="best"*
  - Máxima profundidad del árbol, *max depth="10"*
  - Mínimo de instancias requerido para realizar una separación, *min samples split="2"*
  - Mínimo de instancias requerido en un nodo hoja, *min samples leaf="5"*
- Random Forests:
  - Muestreo aleatorio de las instancias, *bootstrap="True"*
  - Cantidad de árboles independientes, *n stimators="8"*
  - Criterio de separación del árbol, *criterion="entropy"*
  - Máxima cantidad de dimensiones a considerar por separación, *max features="sqrt"*
  - Criterio de separación para las máximas dimensiones consideradas, *splitter="best"*
  - Máxima profundidad del árbol, *max depth="10"*
  - Mínimo de instancias requerido para realizar una separación, *min samples split="2"*
  - Mínimo de instancias requerido en un nodo hoja, *min samples leaf="5"*

Por último, los parámetros de configuración para la búsqueda heurística es un total de 300 iteraciones para RMHC e IMO, en 10 repeticiones. Mientras que el total de elementos en la población de iones en IMO, $P_{ob} = 20$. Para el caso de la Ecuación 3.5 al utilizar RMHC, $alpha = 0.3$.

4.2. Evaluación de los algoritmos de Aprendizaje de Máquina en los sistemas IDS

Los algoritmos de ML son utilizados cómo función de evaluación en los métodos de búsqueda heurística, para ésto se determina aquel algoritmo de los 4 utilizados en el presente trabajo de Tesis, el cual presenta un mejor equilibrio entre porcentaje de exactitud y tiempos de entrenamiento/evaluación. Los tiempos de entrenamiento y evaluación, así cómo porcentaje de

exactitud, son obtenidos mediante el promedio de 10 iteraciones en cada uno de los 4 algoritmos, dichos resultados se muestran en la Tabla 4.1 para el caso de 2 clases, mientras que la Tabla 4.2 muestra los resultados para el caso de 5 clases.

De la Tabla 4.1 es posible observar que los cuatro clasificadores obtienen un aceptable porcentaje de exactitud al ser evaluados únicamente en la situación binaria dónde un registro de conexión es considerado cómo ataque o no. Los tiempos de entrenamiento y evaluación del algoritmo KNN son los más elevados al tener que calcular la distancia de cada una de las instancias con respecto a sus vecinos. En contraste, el algoritmo DT tiene los tiempos más bajos en el entrenamiento y evaluación, sin embargo cómo todos los algoritmos del tipo árbol, tiende a sobreentrenarse al problema lo que provoca que se adecue específicamente a la base de datos de entrenamiento NSLKDD y padezca al detectar ataques los cuales no se encuentran dentro de ésta, es decir, instancias desconocidas específicas en ENSLKDD.

Tabla 4.1: Resultados de los algoritmos de ML aplicados en los sistemas IDS para el caso de 2 clases

Algoritmo	Tiempo de Entrenamiento (s)	Tiempo de Evaluación (s)	% de Exactitud
KNN	92.183	81.192	82.763
SVM's	137.293	12.036	85.779
DT	0.178	0.009	79.775
RF	0.766	0.119	84.762

Mediante la Tabla 4.2 se muestra una disminución en el porcentaje de exactitud de los algoritmos de ML al tener que discernir entre las 5 posibles clases: DoS, U2R, R2L, Probe y Normal. Nuevamente el algoritmo KNN posee los tiempos más altos, mientras que SVM's mantiene el mayor porcentaje de exactitud, sin en cambio, su tiempo de entrenamiento se mantiene cómo el más elevado al tener que crear los hiperplanos para separar las 5 clases.

De las Tablas 4.1 y 4.2, el algoritmo RF pese a no proporcionar el mayor porcentaje de exactitud, a diferencia de SVM's, ni otorgar los menores tiempos de entrenamiento y evaluación en contraste con DT, situación crítica en los sistemas IDS, posee el mejor equilibrio entre dichos parámetros y por lo antedicho es utilizado cómo función de rendimiento en los algoritmos de búsqueda heurística del presente trabajo. La Figura 4.2 muestra la curva ROC del algoritmo RF, mientras que la Figura 4.3 muestra su matriz de confusión binarias para el caso del total de instancias y en unidades de porcentaje. La Figura 4.4 describe sus matrices de confusión para el caso de 5 clases.

Tabla 4.2: Resultados de los algoritmos de ML aplicados en los sistemas IDS para el caso de 5 clases

Algoritmo	Tiempo de Entrenamiento (s)	Tiempo de Evaluación (s)	% de Exactitud
KNN	88.805	74.545	80.745
SVM's	92.963	12.436	82.758
DT	0.161	0.008	77.129
RF	0.556	0.113	81.75

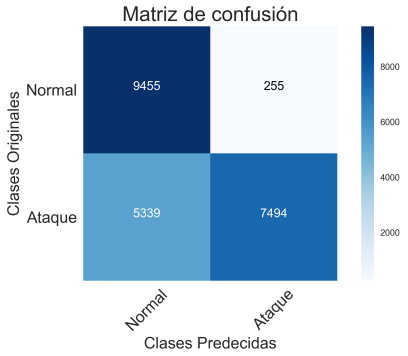
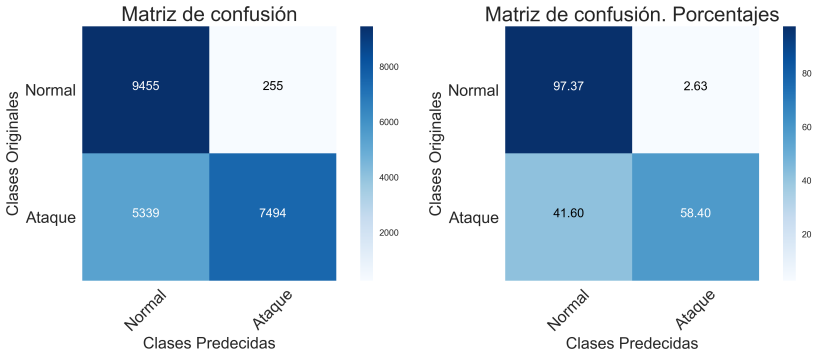


Figura 4.2: Curva ROC del algoritmo Random Forest

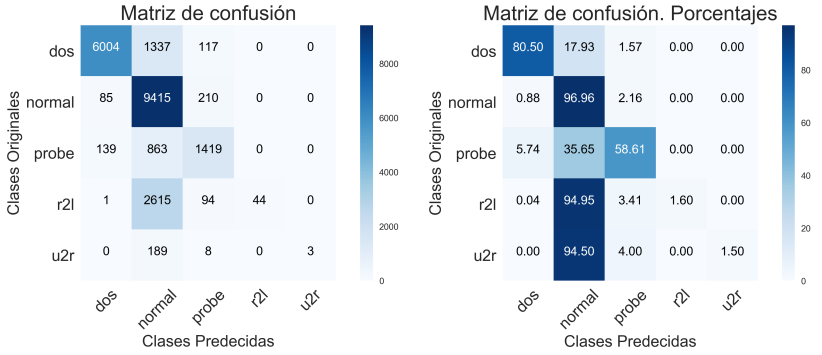


(a) Matriz de confusión binaria del algoritmo RF

(b) Matriz de confusión binaria del algoritmo RF en porcentaje

Figura 4.3: Matrices de confusión binarias del algoritmo RF

.



(a) Matriz de confusión en 5 clases del algoritmo RF

(b) Matriz de confusión en 5 clases del algoritmo RF en porcentaje

Figura 4.4: Matrices de confusión para el caso de 5 clases del algoritmo RF

.

En la Figura 4.4 se observa que RF clasifica de forma correcta aquellos registros de conexión que son considerados del tipo Normal en NSL-KDD99, además la Figura 4.4b muestra un porcentaje de aciertos del 80.5 % para los registros de conexión que pertenecen del tipo de ataque DoS. En contraste, RF sufre al tratar de identificar de manera adecuada los tipos de ataque R2L y U2R, ésto debido a las pocas instancias de entrenamiento de dichos ataques en NSLKDD (Figura 2.8b, Página 31) y al aumento de ellos en ENSLKDD (Figura 2.9b, Página 32). Sumado a ésto los ataques R2L y U2R son, en sus características, muy parecidos a las conexiones del tipo Normal, lo que se puede apreciar en la Figura 2.16 de la Página 45 y por lo tanto resulta muy complejo para RF discernir uno del otro.

4.3. Reducción de dimensiones mediante búsqueda heurística

Cómo se ha hecho mención, la búsqueda heurística necesita una función de evaluación f_{it} la cual determine la calidad de una posición en el espacio de estados. El algoritmo de aprendizaje de máquina RF es utilizado al brindar el mejor equilibrio entre tiempos de entrenamiento y evaluación, además de un buen porcentaje de exactitud en la clasificación de los diferentes ataques de NSL-KDD99. Tanto el algoritmo RMHC cómo IMO son evaluados con un total de 300 iteraciones; RMHC cuenta con una *única* solución del espacio de búsqueda por iteración al ser un algoritmo de trayectoria simple, mientras que IMO cuenta con 20 soluciones del espacio de búsqueda por iteración al ser un algoritmo poblacional y contar con dichos elementos en la población. El proceso de búsqueda heurística es repetido 10 veces, las dimensiones obtenidas son aquellas que aparecen en al menos *22* diferentes repeticiones para

el caso de RMHC y *25* para el caso de IMO

4.3.1. RMHC y SA

La Tabla 4.3 muestra las dimensiones obtenidas en las bases de datos NSLKDD y ENSLKDD las cuales otorgan una reducción mínima de la exactitud de los algoritmos clasificadores a la vez que minimizan drásticamente los tiempos de evaluación y entrenamiento. Dichas dimensiones son encontradas a través del algoritmo RMHC y SA.

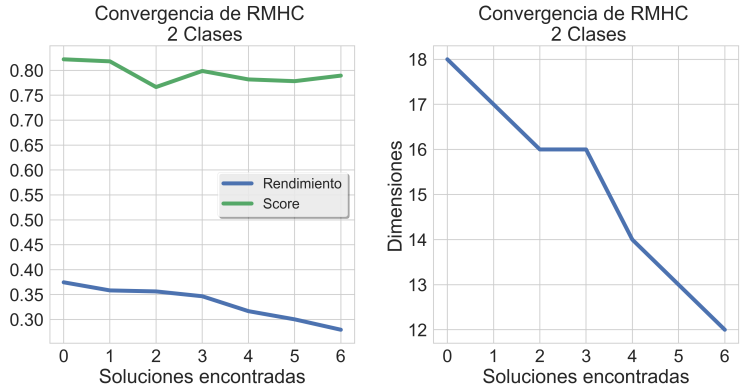
Tabla 4.3: Reducción de dimensiones mediante RMHC y SA, para 2 y 5 clases

Clases	Tiempo (min)	Dimensión / Característica del tráfico de Red	Total
2	35.7164	1,3,4,5,26,31,32,36,39	9
5	36.1573	2,3,5,9,11,20,22,23,27,28,31,34,38	13

Las Figuras 4.5 y 4.6muestran el proceso de convergencia de la exactitud del clasificador cada vez que es encontrada una nueva solución en el espacio de búsqueda la cual fuese mejor a la anterior, es decir, cada vez que el Rendimiento F de la Ecuación 3.5 en alguna iteración fuese menor, ésta nueva solución en el espacio de búsqueda se considera cómo la mejor solución encontrada y es almacenada.

Ésta disminución de F se puede apreciar en las Figuras 4.5a y 4.6a con un valor menor para cada solución encontrada, en contraste, la exactitud del clasificador (score) disminuye de forma mucho más suave perdiendo apenas un ligero porcentaje en comparación con un sistema IDS con dimensiones completas.

Es posible observar además, una disminución abrupta del total de dimensiones por solución encontrada en las Figuras 4.5b y 4.6b, yendo de 18 dimensiones en las primera solución a 12 y de 22 dimensiones a 11, respectivamente.

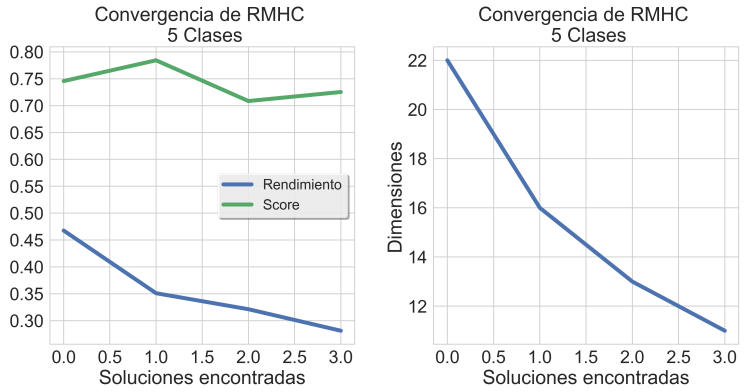


(a) Convergencia del porcentaje de exactitud junto con el total de rendimiento.

(b) Convergencia del total de dimensiones.

Figura 4.5: Convergencia del porcentaje de exactitud, rendimiento R_e y el total de dimensiones utilizando RMHC y SA, en 2 clases.

.



(a) Convergencia del porcentaje de exactitud junto con el total de rendimiento.

(b) Convergencia del total de dimensiones.

Figura 4.6: Convergencia del porcentaje de exactitud, rendimiento R_e y el total de dimensiones utilizando RMHC y SA, en 5 clases.

.

4.3.2. IMO y SA

La Tabla 4.4 muestra las dimensiones obtenidas en las bases de datos NSLKDD y ENSLKDD las cuales otorgan una reducción mínima de la exactitud de los algoritmos clasificadores a la vez que minimizan drásticamente

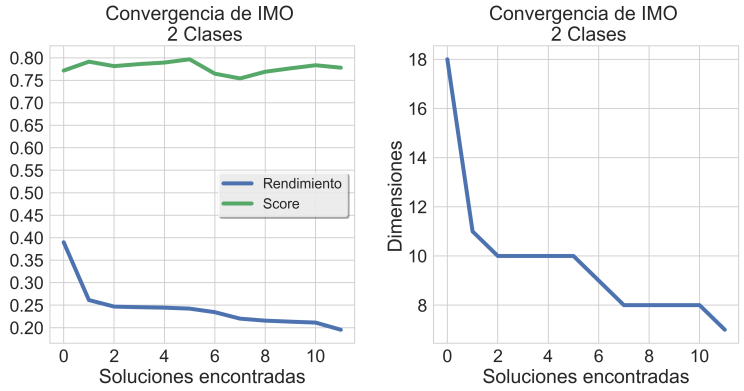
los tiempos de evaluación y entrenamiento. Dichas dimensiones son encontradas a través del algoritmo IMO y SA.

Tabla 4.4: Reducción de dimensiones mediante IMO y SA, para 2 y 5 clases

Clases	Tiempo (H)	Dimensión / Característica del tráfico de Red	Total
2	5.45	2,3,4,15,19,31,39,40	8
5	7.16	3,4,6,7,8,11,17,18,33,37	10

Las Figuras 4.7 y 4.8 muestran el proceso de convergencia de la exactitud del clasificador RF, además de su función de rendimiento R_e por solución encontrada junto con el total de dimensiones. La Figura 4.7 describe el proceso de convergencia del algoritmo IMO para el caso de 2 clases, mientras que la Figura 4.8 para el caso de 5 clases. En La Figura 4.7a se puede apreciar un valor mucho menor de inicio para R_e en las primeras soluciones encontradas, así cómo una disminución drástica conforme avanzan las iteraciones, en contraste con la Figura 4.5a la cual converge de manera más lenta. La causa de lo antedicho es la cualidad del algoritmo IMO de convergir hacia las mejores soluciones del espacio de búsqueda, dónde en cada iteración se busca al mejor elemento entre la población de aniones y cationes, para aproximarse a él a través de los demás elementos de la población. Una vez que los elementos de la población se encuentran lo suficientemente cerca del mejor, se separan nuevamente para volver a explorar el espacio de búsqueda. Es posible observar además en la Figura 4.7b una reducción significativa en la cantidad de dimensiones a diferencia de las presentadas en la Figura 4.5b obtenidas mediante RMHC y SA, pese a que éstas por sí mismas ya poseen una reducción a 8 y 10 dimensiones de las 40 dimensiones originales.

IMO en general ofrece mejores resultados en cuanto al Rendimiento R_e manteniendo el porcentaje de exactitud del clasificador RF con una reducción mínima, a diferencia del proceso de búsqueda heurística del algoritmo RMHC. No obstante, los tiempos de convergencia de IMO al ser un algoritmo del tipo poblacional y ya que el sistema necesita entrenar y evaluar a cada uno de los elementos de la población por iteración, son mucho mayores que los requeridos por RMHC al ser una solución de trayectoria simple.

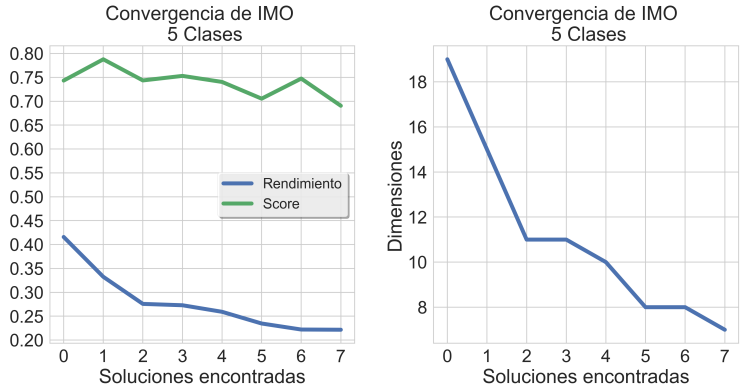


(a) Convergencia del porcentaje de exactitud junto con el total de rendimiento.

(b) Convergencia del total de dimensiones.

Figura 4.7: Convergencia del porcentaje de exactitud, rendimiento R_e y el total de dimensiones utilizando IMO y SA, en 2 clases.

.



(a) Convergencia del porcentaje de exactitud junto con el total de rendimiento.

(b) Convergencia del total de dimensiones.

Figura 4.8: Convergencia del porcentaje de exactitud, rendimiento R_e y el total de dimensiones utilizando IMO y SA, en 5 clases.

.

4.4. Sistema IDS con dimensiones completas y reducidas

La Tabla 4.5 muestra en forma de comparativa los tiempos de entrenamiento, evaluación y porcentaje de exactitud para el sistema IDS utilizando

la base de datos NSL-KDD99 en el caso de 2 y 5 clases, para un sistema utilizando el total de dimensiones (40) y dos sistemas con dimensiones reducidas, el primero obtenido mediante RMHC y SA, y el segundo a través de IMO y SA. Dicha Tabla es obtenida mediante 10 iteraciones utilizando el algoritmo RF en el sistema IDS con dimensiones completas y en los sistemas IDS con dimensiones reducidas.

De la Tabla 4.5 es posible observar una disminución del 26.08 % para el análisis de 2 clases en el tiempo de entrenamiento en el sistema con 40 dimensiones con respecto a RMHC, y un 46.28 % para el caso de IMO. Con respecto al tiempo de evaluación del sistema IDS con dimensiones completas, éste permanece inalterado debido a que el algoritmo RF realiza un mayor costo computacional en la creación de las ramas de cada uno de los árboles, las cuales corresponden a la columna del Tiempo de Entrenamiento (s). Una vez que dichas ramificaciones han sido creadas el procesamiento computacional recae únicamente la capacidad del ordenador dónde el código se ejecute. Paralelamente en el caso del sistema IDS en 2 clases el porcentaje de exactitud con dimensiones completas sufre apenas una disminución del 4.38 % para el caso de RMHC y un 5.96 % para el caso del IMO.

Tabla 4.5: Comparativa del Sistema IDS con dimensiones completas y reducidas.

Clases	Dimensiones	Tiempo de Entrenamiento (s)	Tiempo de Evaluación (s)	Porcentaje de Exactitud
Sistema IDS con dimensiones completas				
2	40	0.7663	0.119	84.76
5	40	0.5565	0.113	81.75
Sistema IDS mediante RMHC y SA				
2	9	0.5657	0.1151	81.05
5	13	0.5677	0.1115	77.50
Sistema IDS mediante IMO y SA				
2	8	0.4710	0.1104	79.03
5	10	0.4743	0.1117	74.67

Los resultados obtenidos en la Tabla 4.5 para el caso de 5 clases, son análogos en 2 clases, es decir, presentan una disminución drástica en los tiempos entrenamiento y mantienen los tiempos evaluación, mientras que propician una disminución mínima en el porcentaje de exactitud, sin embargo, el porcentaje de exactitud para el caso de 5 clases se ve desde el inicio disminuido en comparación con 2 clases, al existir registros de conexión pertenecientes a los ataques R2L y U2R, los cuales en sus características son muy parecidos a las conexiones del tipo Normal, por lo cual el algoritmo RF

se ve afectado al tratar de discernirlos.

Capítulo 5

Conclusiones

What we think or what we know or what we believe is in the end of little consequence. The only thing of consequence is what we do

John Ruskin

En éste trabajo de Tesis fue realizado satisfactoriamente un análisis de la exactitud de los algoritmos de aprendizaje de máquina del tipo clasificadores supervisados, K Nearest Neighbors, Support Vector Machines, Decision Trees y Random Forests, aplicados en la base de datos NSL-KDD99 utilizada en la literatura para la evaluación de los sistemas de red de detección de intrusos basados en anomalías, para un sistema con 40 dimensiones al eliminar la dimensión número 20 ya que no aporta ningún tipo de información al sistema y, paralelamente, fue realizada una comparativa del sistema con 40 dimensiones con dos sistemas con dimensiones reducidas obtenidos mediante búsqueda heurística utilizando los algoritmos Random Mutation Hill Climbing e Ions Motion Optimization, el cual fue modificado para ser utilizado en problemas binarios.
De dicho análisis el algoritmo que otorga el mejor equilibrio entre porcentaje de exactitud y tiempos de entrenamiento, es el clasificador Random Forests, por lo cual fue utilizado cómo función f_{it} embebido en la función $F(D_e)$ en los algoritmos de búsqueda heurística. De la misma manera, tanto el preprocesado de normalización de NSL-KDD99, cómo cada una de las configuraciones de los cuatro algoritmos clasificadores utilizados son descritas en el Capitulo 4, y por lo tanto pueden ser replicables los resultados aquí mostrados, dando solución al problema en la literatura en el cual los trabajos no muestran dichas configuraciones y son imposibles de replicar.

Fue programado y aplicado con éxito el método del tipo envoltura a través de la búsqueda heurística con el objeto de reducir dimensiones, dónde el algoritmo de trayectoria simple RMHC ofreció en general un conjunto de

dimensiones los cuales provocan menores tiempos de entrenamiento/evaluación mientras que, IMO ofrece aún menores tiempos de entrenamiento pero en contraste, posee un menor valor en porcentaje de exactitud. No obstante al ser un algoritmo del tipo poblacional, el tiempo de cómputo en el proceso de reducción de dimensiones mediante el algoritmo IMO fue mucho mayor que el algoritmo RMHC, y la calidad del resultado final no presenta una diferencia significativa. Por lo antedicho la implementación de un algoritmo poblacional cómo IMO, no es recomendada.

Fueron determinadas aquellas características de las conexiones de red (también llamadas dimensiones) las cuales resultan tener un mayor impacto en la exactitud del clasificador; las *características básicas* tales cómo la duración de la conexión, la cantidad de bytes de la fuente de origen junto con los bytes de destino, además de las *características de tráfico basadas en host* son aquellas que aportan mayor información al momento de ser entrenados y evaluados los algoritmos clasificadores.

Es claro que cómo hizo mención J. McHugh [8] la base de datos KDD99 tiene severas deficiencias en la distribución de probabilidad de sus 5 clases, concentrándose particularmente en la clase del tipo Normal y DoS. Pese a las mejoras presentadas por M. Tavallaee [9] las cuales mejoran dicha distribución, y después de haber sido realizado un estudio del aporte de información de cada una de las características del tráfico de red no nominales eliminando aquellas redundantes. Por lo antedicho resulta imposible conseguir resultados mayores incluso al 85 % en la exactitud del algoritmo clasificador, lo cual concuerda con [12] al utilizar una base de datos de evaluación la cual contiene ataques desconocidos para el clasificador de la misma manera que éste trabajo de tesis, y difiere de [18] el cual obtiene resultados mayores al 90 % en la exactitud del clasificador. Todos los resultados en la literatura con un porcentaje de exactitud del algoritmo clasificador mayor al 90 % los cuales hagan uso de la base de datos de entrenamiento NSLKDD y la base de datos de evaluación ENSLKDD, se considera que muestran resultados tendenciosos y no están propiamente planteados al obtener resultados no conseguibles con alguno de los algoritmos de aprendizaje de máquina aquí presentados.

5.1. Trabajos futuros

El análisis de la exactitud de los sistemas de red de detección de intrusos basados en anomalías presentado en éste trabajo de tesis, evalúa únicamente los algoritmos clasificadores con mayor uso en la literatura, no obstante, un análisis más exhaustivo el cual incluya además técnicas de aprendizaje de máquina profundo, es conveniente.

Los sistemas de red de detección de intrusos dependen de la calidad de la base de datos de entrenamiento para que los algoritmos de aprendizaje de máquina logren conseguir un alto porcentaje de exactitud, por lo tanto la

generación de una base de datos adecuada, con una buena distribución de las clases las cuales representan el tráfico de la red, es crítica. Dicha base de datos debe de adecuarse además al tráfico particular de la red a analizar, sin generalizar, con el fin de obtener resultados representativos.

Un análisis exhaustivo de la exactitud de los sistemas de red de detección de intrusos focalizado al tráfico de una red de datos en paticular, partiendo de bases de datos de entrenamiento y evaluación para los algoritmos clasificadores generadas de forma propia sin considerar una base de datos generalizada, resulta el paso más significativo para trabajos futuros.

Bibliografía

La obra humana es colectiva, nada que no sea colectivo es solido ni durable.

Miguel de Unamuno, Niebla

[1] J.P. Anderson. *Computer Security Threat Monitoring and Surveillance.* Fort Washington, 1980.

[2] Like Anderson y D.E. Dening. *An intrusion-detection model*, páginas 222–232. IEEE. Transactions on Software Engineering, 1990.

[3] L.T. Lunt y R.A. Jagannathan. *Prototype real-time intrusion-detection expert system.* IEEE Sympsosium on Security and Privacy, 1998.

[4] L. T. Heberlein, G. V. Dias, K.N. Levitt, y B mukherjee. *A network security monitor.* IEEE Symposium on Security and Privacy, 1990.

[5] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, y K.R. Kendall. *Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation.* DARPA Information Survivability Conference and Exposition, 2000.

[6] W. Lee y Y. Zhang. *Intrusion detection in wireless ad-hoc networks*, páginas 275–283. Proceedings of the Annual International Conference on Mobile Computing and Networking, 6 Edición, 2000.

[7] KDD Cup 1999. Knowledge discovery database. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`, October 2017.

[8] J. McHugh. *Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory*, páginas 262–294. ACM Transactions of Information and System Security, 3 Edición, 2000.

[9] Mahbod Tavallaee, Abrahim Bagheri, Wei Lu, y A. Ghorbani. *A Detailed Analysis of the KDD Cup 99 Dataset.* IEEE Symposioum on Computational Inteligence in Security and Defense Applications (CISDA), 2009.

[10] Maheshkumar Sabhnani y Gursel Serpen. *Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context.* EECS Dept University of Toledo, 2002.

[11] R. O. Duda y P. E. Hart. *Pattern Classification and Scene Analysis*, páginas 20–21. John Wiley and Sons, 2 Edición, 1972.

[12] Muhammad Shakil Pervez y Dewan Md. Farid. *Feature Selection and Intrusion classification in NSL-KDD Cup 99 Dataset Employing SVMs*, páginas 1–6. International Conference on Software, Knowledge, Information Management and Applications, 8 Edición, 2014.

[13] You Chen, Wen-Fa Li, y Xue-Qi Cheng. *Toward Building Lightweight Intrusion Detection System Through Modified RMHC and SVMs*, páginas 83–88. 2007 15th IEEE International Conference on Networks, 2007.

[14] Abdulla Amin Aburomman y Mamun Bin Ibne Reaz. *A survey of intrusion detection systems based on ensemble and hybrid classifiers*, páginas 135–152. Computers and Security, 65 Edición, 2016.

[15] Maheshkumar Sabhnani y Gursel Serpen. *Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection Context.* ECCS Dept University of Toledo, 2002.

[16] H. Günes Kayacik, A. Nur Zincir-Heywood, y Malcom I. Heywood. *Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 Intrusion Detection Datasets*, páginas 12–14. Annual Conference on Privacy, Security and Trust, 3 Edición, 2005.

[17] P. G. Kumar y D. Devaraj. *Network Intrusion Detection using Hybrid Neural Networks*, páginas 563–569. International Conference on Signal Processing, Communications and Networking, 2007.

[18] Adetunmbi A. Olusola, Adeola S. Oladele, y Daramola O. Abosede. *Analysis of KDD'99 Intrusion Detection Dataset for Selection of Releance Features.* Proceedings of the World Congress on Engineering and Computer Science, 1 Edición, 2010.

[19] M. Hossein Ahmadzadegan, Ali Asgar Khorshidvand, y Mahdi Ghalbi Valian. *Low-rate False Alarm Intrusion Detection System with Genetic Algorithm Approach.* International Conference on Knowledge-Based Engineering and Innovation, 2 Edición, 2015.

[20] Muhammad Shakil Pervez y Dewan Md. Farid. *Feature Selection and Intrusion Classification in NSL-KDD cup 99 Dataset Employing SVMs*, páginas 1–6. International Conference on Software, Knowledge, Information Management and Applications, 8 Edición, 2014.

[21] H.M. Tahir, A.M. Said, y N.H. Osman. *Improving K-Means Clustering Using Discretization Technique In Network Intrusion Detection System.* International Conference On Computer And Information Sciences, 3 Edición, 2016.

[22] Preeti Aggarwal y Sudhir Kumar Sharma. *An Empirical Comparision of Classifiers to Analyze Intrusion Detection*, páginas 446–450. International Conference on Advanced Computing and Communication Technologies, 5 Edición, 2015.

[23] Ian H. Witten y Eibe Frank. *Data Mining practial Machine Learning Tols and Techniques.* Elsevier, 2 Edición, 2005.

[24] 27000-2016 ISO/IEC. *Information security management systems - Overview and vocabulary.* 2016.

[25] M.U. Farroq, Muhammad Waseem, Anjum Khairi, y Sadia Mazhar. *A critical Analysis on The Security concerns of Internet of Things (IoT)*, páginas 1–6. International Journal of Computer Applications, 7 Edición, 2015.

[26] Luigi Atzori, Antonio Iera, y Giacomodo Morabito. *The Internet of Things: A survey*, páginas 27–46. Elsevier Computer Networks, 54 Edición, 2010.

[27] Joe Wrigth y Jim Harmening. *Computer and Information Security Handbook*, páginas 257–263. Elsevier, 3 Edición, 2009.

[28] R. Bace y P. Mell. *Intrusion Detection System.* NIST Special Publications, 2001.

[29] M. Yang, D. Chen, y X. Zhang. *Anomaly Detection based on Contiguous Expert Voting Algorithms*, páginas 156–161. ICACIA, 2009.

[30] Symantec. Misuse attack signatures. `https://www.symantec.com/security_response/attacksignatures/`, 2017.

[31] SNORT. Open source intrusion prevention. `https://www.snort.org/`, 2017.

[32] BRO. Network analysis framework. `https://www.bro.org/`, 2017.

[33] Suricata. Open source ids / ips / nsm engine. `https://suricata-ids.org/`, 2017.

[34] G. Vigna, F. Valeur, y D. Balzarotti. *Reducting Errors in the Anomaly-based Detection of Web-based Attacks Through the Combined Analysis of Web Request and SQL Queries*, páginas 205–239. Jornal of computer Security, 17 Edición, 2009.

[35] E. Eskin, A. Arnold, M. Preau, y S. Stolfo. *A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data*. Applications of Data Mining in Computer Society, Kluwer Academic Publishers, 2002.

[36] Amzari Jihadi Ghazali, Waleed Al Nuaimy, y Ali Al Atabi. *Comparison of classification models for NSL-KDD data set for network anomaly detection*. Academic Journal of Science, 4 Edición, 2015.

[37] Amjad Hussain Bhat, Sabyasachi Patra, y Debasish Jena. *Machine Learning Approach for Intrusion Detection on Cloud Virtual Machines*. International Journal of Application or Innovation in Engineering and Management (IJAIEM), 2 Edición, 2013.

[38] Lincoln Laboratories. 1998 darpa intrusion detection evaluation data set. `https://www.ll.mit.edu/ideval/data/1998data.html`, 2017.

[39] Kristopher Robert Kendall. *A database of computer attacks for the evaluation of intrusion detection systems*. Massachusetts Institute of Technology, 1999.

[40] Canadian Institute for Cybersecurity University of New Brunswick. Base de datos nsl-kdd. `http://www.unb.ca/cic/datasets/nsl.html`, 2017.

[41] Mahoney M y Chan P. K. *An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection*, páginas 220–237. International Symposium of Recent Advances in Intrusion Detection, 6 Edición, 2003.

[42] Tamraparni Dasu y Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley, 1 Edición, 2003.

[43] Hadley Wickham. *Tidy Data*. Journal of Statistical Software, 59 Edición, 2014.

[44] Luai Al Shalabi, Zyad Shaaban, y Basel Kasasbeh. *Data Mining: A preprocessing Engine*. Journal of Computer Science, 2006.

[45] Arthur L. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*, páginas 535–554. IBM Journal of Research and Development, 1959.

[46] M.E. Elhamahmy, H.N. Elmahdy, y Imane Saroit. *Impact of Data Set Distinction and Normalization in C5.0 Decision Tree*. Cairo University, Departament of Information Technology, 2009.

[47] James V. Stone. *Principal Component Analysis and Factor Analysis*, páginas 129–135. MIT Press, 2004.

[48] Laurens van der Maaten y Geoffrey Hinton. *Visualizing Data using t-SNE*, páginas 2580–2605. Journal of Machine Learning Research, 9 Edición, 2008.

[49] ScikitLearn. Reducción de dimensiones a través del algoritmo pca. `http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`, 2018.

[50] ScikitLearn. Reducción de dimensiones a través del algoritmo t-sne. `http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`, 2018.

[51] ScikitLearn. Machine learning in python. `http://scikit-learn.org/stable/index.html`, 2018.

[52] Numpy. Fundamental package for scientific computing with python. `http://www.numpy.org/`, 2018.

[53] Laurens van der Maaten. t-sne. `https://lvdmaaten.github.io/tsne/`, 2018.

[54] Gareth James, Daniela Witten, Trevor Hastie, y Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*, páginas 21–22,. Springer, 6 Edición, 2015.

[55] Stuart Russell y Peter Norvig. *Artificial Intelligence A Modern Approach*, páginas 737–757. Pearson, 3 Edición, 2013.

[56] Andreas C. Müller. *Introduction to Machine Learning with Python*. O'Reilly, 2 Edición, 2017.

[57] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly, 2 Edición, 2017.

[58] Eric Kim. Everything about the kernel trick. `http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html`, 2017.

[59] Jordan Michael I y Romain Thibaux. The kernel trick. lecture notes. `https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf`, 2017.

[60] Q. Chen, B. Liu, Z. Zhang, y J.J. Liang. *Problem Definition and Evaluation Criteria for CEC 2015 Special Session and Competition on Bound Constrained Single Objective Computationally Expensive Numerical Optimization*. Nanyang Technological University, 2015.

[61] Alberto García Serrano. *Inteligencia Artificial*. Alfaomega, 1 Edición, 2013.

[62] March H. J. Romanycia y Francis Jeffry Pelletier. *What is a heuristic?*, páginas 47–58. Comput. Intell, 1985.

[63] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, y Clifford Stein. *Introduction To Algorithms*. The MIT Press, 3 Edición, 2009.

[64] Stephanie Forrest y Melanie Mitchell. *Relative Building-Block Fitness and the Building-Block Hypothesis*. Foundatios of Genetic Algorithms, 2 Edición, 1993.

[65] Emile Aarts, Jan Korst, y Wil Michiels. *Search Methodologies*. Springer, 1 Edición, 2005.

[66] S. Kirkpatrick, C.D. Gelatt, y M.P. Vecchi. *Optimization by Simulated Annealing*, páginas 671–680. Science New Series, 220 Edición, 1983.

[67] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, 1 Edición, 1992.

[68] James Kennedy y Russell Eberhart. *Particle Swarm Optimization*. IEEE, 1995.

[69] Behzad Javidy, Abdolreza Hatamlou, y Seyedlai Mirjalil. *Ions motion algorithm for solving optimization problems*, páginas 72–79. Applied Soft Computing, 32 Edición, 2015.

[70] D. H. Wolpert y W. G. Macready. *No free lunch theorems for optimization*, páginas 67–82. IEEE Transactions on Evolutionary Computation, 1 Edición, 1997.

[71] Adetunmbi A. Olusola, Adeola S. Oladele, y Daramola O. Abosede. *Analysis of KDD99 Intrusion Detection Dataset for Selection of Relevance Features*, páginas 162–168. World Congress on Engineering and Computer Science, Int. Assoc. Engn, 2010 Edición, 2010.

[72] Nour Moustafa y Jill Slay. *The significant features of the UNSW-NB15 and the KDD99 data sets for Network Intrusion Detection Systems*, páginas 25–31. International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security. IEEE, 4 Edición, 2015.

[73] Swati Paliwal y Ravindra Gupta. *Denial of Service, Probing and Remote to User (R2L) Attack Detection using Genetic Algorithm*, páginas 57–62. International Journal of Computer Applications, 60 Edición, 2012.

[74] Raimundo Real y J Vargas. The probabilistic basis of jaccard's index of similarity. 1996.

[75] Scikit-Learn. Algoritmo clasificador k-nearest neighbors. http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier, 2018.

[76] Scikit-Learn. Algoritmo clasificador support vector machines. http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC, 2018.

[77] Scikit-Learn. Algoritmo clasificador decision tree. http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifierURL, 2018.

[78] Scikit-Learn. Algoritmo clasificador del tipo de ensamble random forest. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, 2018.

[79] Óscar Azeem Becerril Domínguez. Códigos implementados en éste trabajo de tesis. http://github.com/oscarazeem, 2018.

Parte I

Apéndices

Apéndice A

Código implementado

To ask the right question is harder than to answer it.

Georg Cantor

Resumen: El presente apéndice muestra los diferentes códigos utilizados en el lenguaje de programación Python, aplicados en la base de datos NSL-KDD99. Primero describe la aplicación del preprocesado mediante la normalización en la base de datos. Después muestra la función utilizada para encontrar instancias repetidas. Por último exhibe el código utilizado con el objeto de su visualización.

A.1. Preprocesado

```
#Librerías a utilizar

import pandas as pd
import numpy as np

#Función implementada para crear una representación númerica
#del formato de cadena de algunas dimensiones

def Build_the_index_of_a_dimension(self, dataframe_to_build,
    dataframe_indice):
"""dataframe_to_build, nombre del dataframe a quien se
   desea agregar el indice correspondiente
#Hace uso analogo del método LabelEncoder, sin embargo se
   programo para que el etiquetado fuese
#completamente controlado.
```

```
13  #Recibe el dataframe a agregar los indices contenidos en el
        dataframe indice.
14  #estos vienen de un .csv, comunmente"""
15  import pandas as pd
16  j = 0
17  n = 0
18  # creating the general dataframe->pandas_general
19  # columnas del pandas, para anexarlas al concatenar
20  pandas_general = pd.DataFrame(columns=dataframe_to_build.
        columns)
21  # creando la columna al inicio para no tener problemas al
        usar el concact en el axis 0
22  # se tiene que igualar a cero, debido a que las columnas se
        crean como flotantes
23  # si se crea unicamente la columna vacia:
24  # pandas_temporal['Index'+dataframe_indice.columns[0]]=[]
25  pandas_general['Index' + dataframe_indice.columns[0]] = 0
26
27  for elemento in dataframe_indice[dataframe_indice.columns
        [0]].unique():
28      print("Me encuentro trabajando en :" + elemento)
29      # creating the dataframe filtered.
30      ###
31      # filtra el dataframe original (dataframe_to_build)
            mediante la columna del
32      #"tipo" (protocol, serivce,etc)
33      # despues iguala esa columna con el elemento del for
34      # despues filtra todo el dataframe
35      pandas_temporal = dataframe_to_build[dataframe_to_build[
            dataframe_indice.columns[0]] == elemento]
36      if not (pandas_temporal.empty):
37          # Building the index for pandas_general
38          index_elemento = dataframe_indice[dataframe_indice[
                dataframe_indice.columns[0]] == elemento] \
39              [dataframe_indice.columns[1]].iat[0]
40          # Declaring pandas_temporal
41          # pandas_temporal=dataframe_indice[dataframe_indice.
                TypeOfAttacks==tipeofattack]
42          # slicing the temporal pandas
43          pandas_temporal['Index' + dataframe_indice.columns
                [0]] = index_elemento
44          # concatenaning both, pandas_general and
                pandas_temporal
45          pandas_general = pd.concat([pandas_general,
                pandas_temporal], join='inner', axis=0, )
46          pandas_temporal = None
47          j = j + 1
48      else:
```

```
49          print("\n NO SE encuentra valores para: " + elemento
                + "\n")
50          n = n + 1
51  print("Trabaje: " + str(j) + " veces")
52  print("No encontre valores: " + str(n) + " veces")
53  # sorting the pandas_general
54  # print(pandas_general)
55  pandas_general = pandas_general.sort_index()
56  print("La longitud de la matriz inicial es: " + str(len(
        dataframe_to_build)))
57  print("La longitud de la matriz final es: " + str(len(
        pandas_general)))
58
59  return pandas_general
60
61  #Lectura de la base de datos NSL-KDD99, en su estado RAW
62  test=pd.read_csv("./dataset_labeled/nsl_testing_labeled.csv
        ")
63  tra=pd.read_csv("./dataset_labeled/nsl_training_labeled.csv
        ")
64
65  #Lectura de diccionarios para cada una de
66  #las dimensiones en formato cadena:
67  #Protocol Type (2), Service(3) y Flag (4)
68  #
69  indice_protocolo= pd.read_csv("./dataset_labeled/
        indice_protocol_columna.csv")
70  indice_service=pd.read_csv("./dataset_labeled/
        indice_service_columna.csv")
71  indice_flag=pd.read_csv("./dataset_labeled/
        indice_flag_columna.csv"))
72
73  #"""
74  #Creando las etiquetas, para dichas dimensiones
75  #Etiquetas de Protocolo
76
77  Test=mc.Build_the_index_of_a_dimension(test,indice_protocolo
        )
78  Tra=mc.Build_the_index_of_a_dimension(tra,indice_protocolo)
79
80  #Etiquetas de Servicio
81  Test=mc.Build_the_index_of_a_dimension(Test,indice_service)
82  Tra=mc.Build_the_index_of_a_dimension(Tra,indice_service)
83
84  #Etiquetas de Flag
85  Test=mc.Build_the_index_of_a_dimension(Test,indice_flag)
86  Tra=mc.Build_the_index_of_a_dimension(Tra,indice_flag)
87
88  #Asignando los valores números a las dimensiones
```

```
89  #protocol type
90  Test[Test.columns[1]]=Test[Test.columns[47]]
91  Tra[Tra.columns[1]]=Tra[Tra.columns[47]]
92  #service
93  Test[Test.columns[2]]=Test[Test.columns[48]]
94  Tra[Tra.columns[2]]=Tra[Tra.columns[48]]
95  #Flag
96  Test[Test.columns[3]]=Test[Test.columns[49]]
97  Tra[Tra.columns[3]]=Tra[Tra.columns[49]]
98
99  #Eliminando las columnas recien agregadas
100 Test=Test.drop(['TypeOfAttack', 'TimesApparead', 'Group', '
        IndexAttack', 'IndexByGroup',
101         'IndexBinary', 'IndexProtocolType', 'IndexService', '
                IndexFlag'],axis=1)
102 Tra=Tra.drop(['TypeOfAttack', 'TimesApparead', 'Group', '
        IndexAttack', 'IndexByGroup',
103         'IndexBinary', 'IndexProtocolType', 'IndexService', '
                IndexFlag'],axis=1)
104
105 #Normalizando
106 from sklearn.preprocessing import MinMaxScaler
107 scaler=MinMaxScaler()
108
109 Tra_std=scaler.fit_transform(Tra)
110 Test_std=scaler.transform(Test)
111
112 #Declarando en un dataframe el dataset normalizado
113 Pstd_Tra=pd.DataFrame(Tra_std)
114 Pstd_Test=pd.DataFrame(Test_std)
115 Pstd_Tra.columns=Tra.columns
116 Pstd_Test.columns=Test.columns
117
118 #Almacenando las bases de datos
119 Pstd_Tra.to_csv("./Final_Datasets/MINMAX_FD_NSL_Tra.csv",
        index=False)
120 Pstd_Test.to_csv("./Final_Datasets/MINMAX_FD_NSL_Test.csv",
        index=False)
```

A.2. Identificación de patrones con múltiples etiquetas y redundantes

```
1  def encuentra_filas_iguales(self,dataset_inicio,inicio='
       Duration',fin='Group'):
2  import pandas as pd
3  import numpy as np
```

```
4  from sklearn import preprocessing
5  dataset = dataset_inicio.loc[:, inicio:fin]
6  columnas = dataset.columns
7  print("***PREPROCESSING***")
8  for indice in range(0, len(columnas)):
9      if type(dataset.loc[0, columnas[indice]]) == str:
10         print('Building the labels dimension number= %d,
               perteneciente al Nombre: %s' %(
11         indice, columnas[indice]))
12         # label enconder, object
13         label_encoder = preprocessing.LabelEncoder()
14         # label encoer, fit
15         label_encoder.fit(dataset[columnas[indice]])
16         # building the dataframe
17         dataset[columnas[indice]] = label_encoder.transform(
               dataset[columnas[indice]])
18     # for donde multiplica todas las columnas, para evitar
           los iguales al sumar
19     dataset[columnas[indice]] = dataset[columnas[indice]] *
           (np.random.randn())
20 # obteniendo la suma del dataset
21 suma = dataset.sum(axis=1)
22 # adding the "suma" series to the dataset_inicio
23 dataset_inicio["suma"] = suma
24 # borrando los duplicados
25 series_suma = dataset_inicio[dataset_inicio.suma.duplicated(
       keep=False)]
26 # obteniendo los valores repetidos mediante group by
27 series_suma = dataset_inicio.groupby('suma').apply(lambda x:
       list(x.index))
28 #
29 print("***VALORES REPETIDOS***")
30 print("Existen %d valores repetidos" %(len(dataset_inicio)
       - len(series_suma)))
31 # arriba todo bien
32 for indice in range(0, len(series_suma)):
33     a = []
34     if len(series_suma[series_suma.index.values[indice]]) >
           1:
35         print("***NUEVA REPETICION en el indice suma: %f" %
               (series_suma.index.values[indice]))
36         print(
37             "Tiene un total de instancias iguales a: %d" %(
                   len(series_suma[series_suma.index.values[
                   indice]])))
38         # aqui parece el problema
39         for cantidad_instancias_repetidas in range(0, len(
               series_suma[series_suma.index.values[indice]])):
```

```
40                print("No de instancia: %d que equivale al
                      indice dataset: %d" %(
                      cantidad_instancias_repetidas,
41                series_suma[series_suma.index.values[indice]][
                      cantidad_instancias_repetidas]))
42                ataque = dataset_inicio.loc[
43                    series_suma[series_suma.index.values[indice
                          ]][cantidad_instancias_repetidas], "
                          TypeOfAttack"]
44                clase = dataset_inicio.loc[
45                    series_suma[series_suma.index.values[indice
                          ]][cantidad_instancias_repetidas], "
                          Group"]
46                print("Pertenece al tipo de ataque: %s de la
                      clase %s" %(ataque, clase))
47                a.append(series_suma[series_suma.index.values[
                      indice]][cantidad_instancias_repetidas])
48                if cantidad_instancias_repetidas == (len(
                      series_suma[series_suma.index.values[indice
                      ]])-1):
49                    with pd.option_context('display.max_rows',
                          None, 'display.max_columns', 42):
50                        print(dataset_inicio.loc[a, '
                              TypeOfAttack':])
```

A.3. Visualización de NSLKDD y ENSLKDD

```
1  #Lectura de Librerías
2  print(__doc__)
3  from time import time
4  import numpy as np
5  import pandas as pd
6  import matplotlib.pyplot as plt
7  from matplotlib import offsetbox
8  from sklearn import (manifold, datasets, decomposition,
       ensemble, discriminant_analysis, random_projection)
9  import time
10
11 #Lectura de las bases de datos de NSL-KDD99
12 #Nsl learning
13 nsl_training=pd.read_csv('./Final_Datasets/FD_NSL_Tra.csv')
14 #Nsl testing
15 nsl_testing=pd.read_csv('./Final_Datasets/_FD_NSL_Test.csv')
16
17 #Lectura de las etiquetas de NSL-KDD99
18 #NSL TESTING LABELS
```

```
19 #Index binary
20 L_nsl_testing_indexbinary=pd.read_csv("./Final_Datasets/
       L_nsl_testing_indexbinary.csv")
21 #Index by Group
22 L_nsl_testing_indexbygroup=pd.read_csv("./Final_Datasets/
       L_nsl_testing_indexbygroup.csv")
23 #Index tipe of attack
24 L_nsl_testing_indextipeofattack=pd.read_csv("./
       Final_Datasets/L_nsl_testing_indextipeofattack.csv")
25
26
27 #NSL TRAINING LABELS
28 L_nsl_training_indexbinary=pd.read_csv("./Final_Datasets/
       L_nsl_training_indexbinary.csv")
29 #Index by Group
30 L_nsl_training_indexbygroup=pd.read_csv("./Final_Datasets/
       L_nsl_training_indexbygroup.csv")
31 #Index tipe of attack
32 L_nsl_training_indextipeofattack=pd.read_csv("./
       Final_Datasets/L_nsl_training_indextipeofattack.csv")
33
34
35 #Declaración del objeto T-SNE
36 tsne = manifold.TSNE(n_components=2, init='pca',
       random_state=0)
37
38 #Leyendo un segmento pequeño de NSL-KDD99
39 total=3500
40 X=nsl_training.values
41 np.random.seed(8)
42 indice=np.random.choice(list(range(0,X.shape[0])), total)
43 X=X[indice,:]
44
45 #Visualización en 2 clases
46 #Lectura de las Labels binarias
47 Labels=L_nsl_training_indexbinary.values.T
48 Labels=Labels[0,indice]
49
50 #2 clases
51 plt.rcParams['figure.figsize']= 6.2,6.2
52 sns.set_style('whitegrid')
53
54 plt.scatter(Database[Labels==0][:,0], Database[Labels
       ==0][:,1], label='No ataque', c='blue',
55             marker='s')
56 plt.scatter(Database[Labels==1][:,0], Database[Labels
       ==1][:,1], label='Ataque', c='red')
57
58 plt.legend(prop={'size':15},frameon=True,shadow=True)
```

```
59  plt.xticks(fontsize=20)
60  plt.yticks(fontsize=20)
61  plt.title('NSLKDD. Clasificación Binaria',fontsize=22)
62
63  #ax.yaxis.label.set_size(40)
64  plt.savefig("./TSNE_nsl_2_classes.png", dpi=300)
65  plt.show()
66
67  #Visualización en 5 clases
68  #Lectura de las labels en 5 clases
69  Labels=L_nsl_training_indexbygroup.values.T
70  Labels=Labels[0,indice]
71
72  #5 clases
73  plt.rcParams['figure.figsize']= 6.2,6.2
74  plt.scatter(Database[Labels==1][:,0], Database[Labels
        ==1][:,1], label='Dos', c='blue',marker='s')
75  plt.scatter(Database[Labels==2][:,0], Database[Labels
        ==2][:,1], label='Normal', c='red',marker='d')
76  plt.scatter(Database[Labels==3][:,0], Database[Labels
        ==3][:,1], label='Probe', c='sienna',marker='o')
77  plt.scatter(Database[Labels==4][:,0], Database[Labels
        ==4][:,1], label='R2L', c='green',marker='x')
78  plt.scatter(Database[Labels==5][:,0], Database[Labels
        ==5][:,1], label='U2R', c='turquoise',marker='v')
79
80  plt.legend(prop={'size':15},frameon=True,shadow=True)
81  plt.xticks(fontsize=20)
82  plt.yticks(fontsize=20)
83  plt.title('NSLKDD. 5 Clases Principales',fontsize=22)
84  plt.savefig("TSNE_nsl_5_classes.png", dpi=300)
85  plt.show()
```

Apéndice B

Código Búsqueda Heurística

Pones tu pie en el camino y si no cuidas tus pasos, nunca sabes a donde te pueden llevar.

John Ronald Reuel Tolkien, El Señor de los Anillos

Resumen: El presente apéndice muestra los códigos implementados en el lenguaje de programación Python para la realización de la búsqueda heurística mediante el método del tipo trayectoria simple Random Hill Mutation Climbing y el método poblacional Ions Motion Optimization.

B.1. Random Hill Mutation Climbing

```
#
resultados_score=[]
resultados_cantidad_dimensiones=[]
resultados_rendimiento=[]
resultados_dimensiones_activas=[]
resultados_indices_dimensiones_activas=[]

#while
iteracion_general=0
start_time = time.time()
while iteracion_general<10:
    i=0
    M=38
    iteraciones_maximas=300
```

```
15      #la declaracion tiene que ser 40! cómo puede tener menos
            dimensinoes???!!!
16      mejor_solucion=np.random.randint(2, size=40)
17      #almacena los score de todas las iteraciones
18      registro_score_iteracion=[]
19      #almacena la probabilidad de error de todas las
            iteraciones
20      registro_perror_iteracion=[]
21      #almacena el vector binario de todas las iteraciones
22      registro_dimensiones_iteracion=[]
23      #registro de la probabilidad de error de los valores
            aceptados
24      registro_perror_valores_aceptados=[]
25      #almacena unicamente el score de los vectores binarios
            aceptados
26      registro_score_valores_aceptados=[]
27      #almacena el vector binario de los valores aceptados (
            dimensiones_activas)
28      #es un sinonimo de la mejor solucion del espacio de
            búsqueda
29      registro_dimensiones_activas_valores_aceptados=[]
30      #registro de las dimensiones activas por iteracion
31      registro_dimensiones_activas_iteracion=[]
32      #registro de la cantidad de dimensiones activas por
            iteracion
33      registro_cantidad_dimensiones_activas_iteracion=[]
34      #registro cantidad de dimensiones activas de los valores
            aceptados
35      registro_cantidad_dimensiones_activas_aceptados=[]
36      #registro indices de las dimensiones activas
37      registro_indices_dimensiones_activas=[]
38
39      #registro del rendimiento por iteracion
40      registro_rendimiento_iteracion=[]
41      #registro del rendimiento de los valore aceptados
42      registro_rendimiento_aceptados=[]
43
44
45
46      print("solucion_inicial")
47      print(mejor_solucion)
48      print("\n")
49
50
51      #evaluacion de los primeros parametros para la PRIMER
            solucion
52      #la cual se considera la mejor
53      mejor_solucion=np.random.randint(2, size=40)
54
```

```
55
56          #dimensiones temporales se omite porque YA genera el
                siguiente vector.
57          [dimensiones_temporales,zeros,unos,training,testing]=
                obtencion_parametros_binarios(mejor_solucion,M)
58
59          #craeción del modelo para el caso de DOS CLASES
60          modelo.fit(training,L_nsl_training_indexbygroup.values.
                ravel())
61          predicciones=modelo.predict(testing)
62          mejor_score=accuracy_score(L_nsl_testing_indexbygroup,
                predicciones)
63          mejor_perror=1-mejor_score
64          mejor_rendimiento=funcion_rendimiento(mejor_perror,unos,
                N=40,alfa=.3)
65
66
67
68          registro_score_valores_aceptados.append(mejor_score)
69          registro_perror_valores_aceptados.append(mejor_perror)
70          registro_dimensiones_activas_valores_aceptados.append(
                mejor_solucion)
71          registro_cantidad_dimensiones_activas_aceptados.append(
                np.array(unos).shape[1])
72          registro_rendimiento_aceptados.append(mejor_rendimiento)
73          registro_indices_dimensiones_activas.append(unos)
74
75          while (i<iteraciones_maximas):
76              print("\n\n Iteracion: %s \n\n" %(i+1))
77
78              #desde el inicio genera el siguiente vector binario
                    partiendo que el mejor fue el primero generado.
79              #dimensinoes temporales es el vector binario que va
                    mutando con cada iteración
80              [dimensiones_temporales,zeros,unos,training,testing
                    ]=obtencion_parametros_binarios(
                    dimensiones_temporales,M)
81              #Se evaluan los resultados
82              #se entrena el modelo
83              modelo.fit(training,L_nsl_training_indexbygroup.
                    values.ravel())
84              #se obtienen las predicciones
85              predicciones=modelo.predict(testing)
86
87              #se obtiene el accuracy score
88              score_actual=accuracy_score(
                    L_nsl_testing_indexbygroup,predicciones)
89              #se obtiene la probabilidad de error
90              perror_actual=1-score_actual
```

```
91          #se obtiene el rendimiento de dicha solucion
92          rendimiento_actual=funcion_rendimiento(perror_actual
                ,unos,N=40,alfa=.3)
93
94
95
96          #se declara un vector que contiene todos los score
                de todas las iteraciones
97
98          #Se almacenan los resultados
99          registro_score_iteracion.append(score_actual)
100         registro_dimensiones_iteracion.append(
                dimensiones_temporales)
101         registro_perror_iteracion.append(perror_actual)
102         registro_dimensiones_activas_iteracion.append(unos)
103         registro_cantidad_dimensiones_activas_iteracion.
                append(np.array(unos).shape[1])
104         registro_rendimiento_iteracion.append(
                rendimiento_actual)
105
106
107         #aqui entra el If
108         if rendimiento_actual < mejor_rendimiento:
109             #se actualiza la mejor solucion del espacio de
                    búsqueda
110             mejor_solucion=np.array(dimensiones_temporales)
111             print("\n Se encontró una mejor solución \n")
112             print("Score mejor anterior: %s" %mejor_score)
113             mejor_score=score_actual
114             print("Score mejor actual: %s" %mejor_score)
115             print("Rendimiento anterior: %s" %
                    mejor_rendimiento)
116             print("Rendimiento actual: %s" %
                    rendimiento_actual)
117             mejor_rendimiento=np.array(rendimiento_actual)
118
119             print("Cantidad de Dimensiones activas
                    anteriores: %s" %(
120                 registro_cantidad_dimensiones_activas_aceptados
                        [-1]))
121             #np.array(registro_dimensiones_activas_iteracion
                    [-1]).shape[1]
122             print("Cantidad de Dimensiones activas actuales:
                     %s" %np.array(unos).shape[1])
123
124
125
126
127             #impresion de parametros
```

```
128                 registro_score_valores_aceptados.append(
                        mejor_score)
129                 registro_perror_valores_aceptados.append(
                        perror_actual)
130                 registro_dimensiones_activas_valores_aceptados.
                        append(dimensiones_temporales)
131                 registro_rendimiento_aceptados.append(
                        rendimiento_actual)
132                 registro_cantidad_dimensiones_activas_aceptados.
                        append(np.array(unos).shape[1])
133                 registro_indices_dimensiones_activas.append(unos
                        )
134
135
136             #Se imprimen los resultados
137             print("Indice de las Dimensiones activas: ")
138             print(unos)
139             print("Total de dimensiones activas: %s" %np.array(
                    unos).shape[1])
140             print("Porcentaje de exactitud: %s (accuracy score)"
                     %accuracy_score(L_nsl_testing_indexbygroup,
                    predicciones))
141             print("Probabilidad de error (Perror): %s" %(1-
                    accuracy_score(L_nsl_testing_indexbygroup,
                    predicciones)))
142             print("Rendimiento (RE): %s" %rendimiento_actual)
143
144
145
146
147
148             #M se calcula al final
149
150             #M=M_dimensiones_a_mutar_2(i,iteraciones_maximas,
                    M_Max=40)
151
152             M=M_dimensiones_a_mutar(perror_actual,i,
                    iteraciones_maximas,M_Max=40)
153
154
155             #M debe ser entero o da un error al elegir el
                    sampleo aleatorio en la seleccion de dimensiones
156             #problema porque M es flotante
157             M=int(M)
158
159
160             i=i+1
161
```

```
162         resultados_score.append(registro_score_valores_aceptados
                )
163         resultados_cantidad_dimensiones.append(
                registro_cantidad_dimensiones_activas_aceptados)
164         resultados_rendimiento.append(
                registro_rendimiento_aceptados)
165         resultados_dimensiones_activas.append(
                registro_dimensiones_activas_valores_aceptados)
166         resultados_indices_dimensiones_activas.append(
                registro_indices_dimensiones_activas)
167 
168         iteracion_general=iteracion_general+1
169     t_entrenamiento_f_o=(time.time() - start_time)
170 
171 
172     print("\n Tiempo de RMHC\n")
173     print("--- %s seconds ---" %t_entrenamiento_f_o)
```

B.2. Ions Motion Optimization

```
1   #Registros de resultados GLOBALES
2   resultados_score=[]
3   resultados_cantidad_dimensiones=[]
4   resultados_rendimiento=[]
5   resultados_dimensiones_activas=[]
6   resultados_indices_dimensiones_activas=[]
7 
8   #while
9   iteracion_general=0
10  maxima_cantidad_de_corridas=5
11  start_time = time.time()
12 
13  while iteracion_general<maxima_cantidad_de_corridas:
14      print("\n\n Corrida %s de: %s" %(iteracion_general,
            maxima_cantidad_de_corridas))
15 
16      i=0
17      M=38
18      iteraciones_maximas=100
19 
20 
21      #REGISTRO DE RESULTADOS POR EJECUCION DE EJERCICIO (300
            iteraciones)
22 
23      #registro de la probabilidad de error de los valores
            aceptados
```

```
24      registro_perror_valores_aceptados=[]
25      #almacena unicamente el score de los vectores binarios
            aceptados
26      registro_score_valores_aceptados=[]
27      #almacena el vector binario de los valores aceptados (
            dimensiones_activas)
28      #es un sinonimo de la mejor solucion del espacio de
            búsqueda
29      registro_dimensiones_activas_valores_aceptados=[]
30      #registro cantidad de dimensiones activas de los valores
             aceptados
31      registro_cantidad_dimensiones_activas_aceptados=[]
32      #registro del rendimiento de los valore aceptados
33      registro_rendimiento_aceptados=[]
34      #registro de los unos
35      registro_indices_dimensiones_activas=[]
36
37      #1) Declaración de parámetros específicos para IMO
38
39
40      #INDICES
41      m_c=0 #mejor cation
42      m_a=0 #menor anion
43      m_m=0 #mejor elemento entre aniones y cationes
44      m_e=0 #mejor elemento hasta el momento entre todas las
            iteraciones (best)
45
46
47      #Declaracion de la poblacion de forma aleatoria
48
49      A=np.random.randint(2, size=(10,40))
50      C=np.random.randint(2, size=(10,40))
51
52      #Matriz Dimensiones activas (lista) (dimensiones unos):
53      #dimensiones_activas_aniones=[]
54      #dimensiones_activas_cationes=[]
55
56      I_A=[]
57      I_C=[]
58      I_M=[]
59
60      #Matriz de evaluacion con todos los parámetros
61      M_E_A=np.zeros([10,8])
62      M_E_C=np.zeros([10,8])
63
64
65
66      #####--#####--####
67      #almacena UNICAMENTE el mejor entre ANION y CATION
```

```
68      #Cuando se cumple la condicion que el nuevo ANION/CATION
69      #da un rendimiento MENOR al mejor anterior
70
71
72      #Evaluar el primer elemento, y guardarlo, asumiento que
            el primero es el mejor
73      #PRIMER ELEMENTO
74
75
76      #evaluar todo->guardarlo->aplicar fase liquida/solida (
            para que despues sea vuelto a evaluar)
77
78      #Evaluación Sencilla
79      #Población Aniones
80      [M_E_A,I_A]=evaluacion_sencilla(A,M_E_A,I_A,modelo)
81      #Poblacion Cationes
82      [M_E_C,I_C]=evaluacion_sencilla(C,M_E_C,I_C,modelo)
83
84
85      #encontrando los individuos
86      m_a=mejor_individuo(M_E_A)
87      m_c=mejor_individuo(M_E_C)
88
89      #Evaluación compuesta
90      #Poblacion Aniones
91      [M_E_A]=evaluacion_compuesta(A,C[m_c],M_E_A)
92      #Poblacion Cationes
93      [M_E_C]=evaluacion_compuesta(C,A[m_a],M_E_C)
94
95      #Encontrando al mejor de las dos poblaciones
96
97      if (M_E_A[m_a,5] < M_E_C[m_c,5]):
98          #el anion es mejor solucion
99          m_m=m_a
100         #M_M_T_R=mejor poblacion temporal (matriz de
                resultados)
101         #M_M_T=mejor poblacion temporal
102         M_M_T=np.array(A)
103         M_M_T_R=np.array(M_E_A)
104     else:
105         #el cation es la mejor solucion
106         m_m=m_c
107         M_M_T=np.array(C)
108         M_M_T_R=np.array(M_E_C)
109
110     #almacenando los valores
111
112
113     #Se usa M_M_T_R porque son los RESULTADOS
```

```
114         #5->rendimiento
115         registro_rendimiento_aceptados.append(M_M_T_R[m_m,5])
116         #0->score
117         registro_score_valores_aceptados.append(M_M_T_R[m_m,0])
118         #1->perror
119         registro_perror_valores_aceptados.append(M_M_T_R[m_m,1])
120         #Nuevo mejor elemento de ambas poblaciones (total de
                dimensiones)
121         #M_M_T porque es la mejor poblacion, anion o cation
122         registro_dimensiones_activas_valores_aceptados.append(
                M_M_T[m_m])
123         #Registro de la cantidad de unos
124         #se obtienen los indices del los unos (dimensiones
                activas)
125         registro_indices_dimensiones_activas.append(np.where(
                M_M_T[m_m]==1))
126         #se obtiene el total de dimensiones activas
127         registro_cantidad_dimensiones_activas_aceptados.append(
                np.array(np.where(M_M_T[m_m]==1)).shape[1])
128
129
130         #aplicando la fase líquida en la primer evaluación
131
132         [A,M_E_A]=fase_liquida(A,C[m_c],M_E_A,i,
                iteraciones_maximas)
133         #cationes
134         [C,M_E_C]=fase_liquida(C,A[m_a],M_E_C,i,
                iteraciones_maximas)
135
136
137         #evalua de forma general
138
139
140         while (i<iteraciones_maximas):
141             #Encontrando los criterios de evaluación
142             criterio_A=np.where(M_E_A[:,4]>=.6)
143             criterio_A=np.concatenate(criterio_A).ravel().tolist
                    ()
144             criterio_C=np.where(M_E_A[:,4]>=.6)
145             criterio_C=np.concatenate(criterio_C).ravel().tolist
                    ()
146
147
148             #evaluando las poblaciones
149
150             #Evaluación Sencilla
151             #Población Aniones
152             [M_E_A,I_A]=evaluacion_sencilla(A,M_E_A,I_A,modelo)
153             #Poblacion Cationes
```

```
154             [M_E_C,I_C]=evaluacion_sencilla(C,M_E_C,I_C,modelo)
155
156
157             #encontrando los individuos
158             m_a=mejor_individuo(M_E_A)
159             m_c=mejor_individuo(M_E_C)
160
161             #Evaluación compuesta
162             #Poblacion Aniones
163             [M_E_A]=evaluacion_compuesta(A,C[m_c],M_E_A)
164             #Poblacion Cationes
165             [M_E_C]=evaluacion_compuesta(C,A[m_a],M_E_C)
166
167
168             #Aquí se almacenan los resultados. Antes
169             #De que resulten ser mutados nuevamente. por
170             #las fases líquida/solida
171
172             #Encuentra al mejor entre las dos poblaciones
173
174
175             if (M_E_A[m_a,5] < M_E_C[m_c,5]):
176                 #el anion es mejor solucion
177                 m_m=m_a
178                 #M_M=mejor poblacion temporal
179                 M_M_T=np.array(A)
180                 M_M_T_R=np.array(M_E_A)
181
182             else:
183                 #el cation es la mejor solucion
184                 m_m=m_c
185                 M_M_T=np.array(C)
186                 M_M_T_R=np.array(M_E_C)
187
188
189             #if el nuevo mejor tiene un rendimiento mejor que el
                    mejor anterior
190             #actualiza y salva todos los nuevos valores
191             if M_M_T_R[m_m,5]<registro_rendimiento_aceptados
                    [-1]:
192
193
194                 #Se usa M_M_T_R porque son los RESULTADOS
195                 #5->rendimiento
196                 registro_rendimiento_aceptados.append(M_M_T_R[
                        m_m,5])
197                 #0->score
198                 registro_score_valores_aceptados.append(M_M_T_R[
                        m_m,0])
```

```
199                 #1->perror
200                 registro_perror_valores_aceptados.append(M_M_T_R
                        [m_m,1])
201                 #Nuevo mejor elemento de ambas poblaciones (
                        total de dimensiones)
202                 #M_M_T porque es la mejor poblacion, anion o
                        cation
203                 registro_dimensiones_activas_valores_aceptados.
                        append(M_M_T[m_m])
204                 #Registro de la cantidad de unos
205                 #se obtienen los indices del los unos (
                        dimensiones activas)
206                 registro_indices_dimensiones_activas.append(np.
                        where(M_M_T[m_m]==1))
207                 #se obtiene el total de dimensiones activas
208                 registro_cantidad_dimensiones_activas_aceptados.
                        append(np.array(np.where(M_M_T[m_m]==1)).
                        shape[1])
209
210             #else:
211             #sino es mejor, discrimina esa ultima solucion
212
213
214
215             #IF para entrar en la fase líquida o sólida
216
217             if ((int(len(criterio_A)) <5) or (int(len(criterio_C
                    )) <5)):
218                 #entra en FASE LÍQUIDA
219                 #evaluación
220                 #aniones
221                 [A,M_E_A]=fase_liquida(A,C[m_c],M_E_A,i,
                        iteraciones_maximas)
222                 #cationes
223                 [C,M_E_C]=fase_liquida(C,A[m_a],M_E_C,i,
                        iteraciones_maximas)
224             else:
225                 #entra en Fase Sólida
226                 #aniones
227                 [A,M_E_A]=fase_solida(A,C[m_c],M_E_A,i,
                        iteraciones_maximas)
228                 [C,M_E_C]=fase_solida(C,A[m_a],M_E_C,i,
                        iteraciones_maximas)
229
230
231         #iteraciones, por ejercicio
232
233             i=i+1
234
```

```
235
236
237         #FIN corrida, se almacenan los resultados de dicha
                corrida.
238
239         resultados_score.append(registro_score_valores_aceptados
                )
240         resultados_cantidad_dimensiones.append(
                registro_cantidad_dimensiones_activas_aceptados)
241         resultados_rendimiento.append(
                registro_rendimiento_aceptados)
242         resultados_dimensiones_activas.append(
                registro_dimensiones_activas_valores_aceptados)
243         resultados_indices_dimensiones_activas.append(
                registro_indices_dimensiones_activas)
244
245         iteracion_general=iteracion_general+1
246     t_entrenamiento_f_o=(time.time() - start_time)
247
248
249     print("\n Tiempo de IMO\n")
250     print("--- %s seconds ---" %t_entrenamiento_f_o)
```

Apéndice C

Artículos y Ponencias

The art of doing mathematics is finding that special case that contains all the germs of generality

David Hilbert

Resumen: El presente apéndice muestra los artículos y ponencias los cuales fueron obtenidos mediante el seguimiento de éste trabajo de Tesis.

Algoritmo IMO y SVMs para la elaboración de un sistema ligero IDS basado en anomalías mediante la base de datos NSL-KDD

Becerril Domínguez Óscar Azeem[1], Menchaca García Felipe Rolando[1]
[1]Instituto Politécnico Nacional, Escuela Superior de Ingeniería Mecánica y Eléctrica, Sección de Estudios de Posgrado e Investigación, Unidad Profesional Adolfo López Mateos "Zacatenco", Edificio 5 2do. Piso, Col. Lindavista, C.P. 07300, Ciudad de México
e-mail: oscar.azeem@me.com, fmenchac@gmail.com

***Resumen*—Muchos trabajos se han realizado para la realización de Sistemas de Red de Detección de Intrusos basados en anomalías, mayoría de los cuales hacen uso de la base de datos KDD99 para el entrenamiento de los algoritmos de aprendizaje de máquina. La base de datos KDD99 en tiempos recientes ha sido expuesta por sus múltiples deficiencias tales cómo sus múltiples instancias redundantes, siendo demeritados los resultados que se han obtenido con ella. Para solventar ésta problemática en éste trabajo se hace uso de la base de datos NSL-KDD la cual resuelve la mayoría de los problemas inherentes en KDD99 ofreciendo resultados confiables. NSL-KDD junto con KDD99 cuentan con 41 características del tráfico de red proveniente de una red militar, provocando un alto tiempo de entrenamiento de los algoritmos de aprendizaje de máquina. Con el fin de mejorar los tiempos de aprendizaje de los algoritmos de aprendizaje de máquina, en éste trabajo se hace uso del algoritmo Ions Motion Algorithm cómo estrategia de búsqueda heurística, junto con el algoritmo Support Vector Machines cómo criterio de evaluación con el fin de reducir la cantidad de características propias de NSL-KDD. Los experimentos realizados muestran que es posible entrenar un sistema de red de detección de intrusos aún con una buena precisión, y aumentar de forma notable los tiempos de entrenamiento mediante un sistema de características reducidas usando el algoritmo Ions Motion Optization y Support Vector Machines.**

***Palabras Clave*—Aprendizaje de Máquina, Búsqueda Heurística, IMO, SVM, KDD99, NSL-KDD, IDS**

***Abstract*— Many papers have been done for the realization of Network Intrusion Detection Systems based on anomalies, plenty of them made use of the KDD99 database in order to train the Machine Learning Algorithms. The KDD99 database in recent times has been exposed due to it's many deficiencies such as redundant instances, being discredited the results obtained with it. To solve this problem in this work we use the NSL-KDD dataset, which solves almost all the inherent problems in KDD99, offering reliable results. Both KDD99 and NSL-KDD have 41 features corresponding to the simulated LAN traffic from a military network, triggering a high training time in the Machine Learning Algorithms. With the purpose of improving the learning times of the machine learning algorithms, in this work is used an algorithm called Ions Motion Optimization as heuristic search along with the Support Vector Machines algorithm as evaluation criterion with the end of reduce the amount of features in NSL -KDD. The experiments have shown that it is possible to train an Intrusion Detection System with good accuracy, and improve noticeably the learning times using a lightweight system with reduced features, by the Ions Motion Optimization algorithm and Support Vector Machines.**

***Keywords*— Machine Learning, Heuristic Search, IMO, SVM, KDD99, NSL-KDD, IDS**

I. Introducción

I-A. Sistemas de detección de intrusos

Los sistemas de detección de intrusos (IDS) son una alternativa dinámica de seguridad para proteger la estabilidad, confiabilidad, y disponibilidad de sistemas conectados a Internet [1]. Son clasificados típicamente en dos vertientes, basados en host, Host Intrusion Detection Systems (HIDS) o en red, Network Intrusion Detection Systems (NIDS) [2]. Los sistemas HIDS monitorean los recursos de disco y logs, mientras que los sistemas NIDS se encargan de monitorear el tráfico que pasa a través de la red. Estos sistemas IDS pueden ser desplegados para el análisis de información de la siguiente manera:

- Anomalías: se despliegan algoritmos de Machine Learning (ML) entrenados en alguna base de datos (dataset) constituyente al tráfico que pasa por una red. Los algoritmos ML se encargan de aprender del comportamiento normal de la red, y detectar una desviación de los patrones comunes cómo ataque.
- Misuse: se encargan de monitorear recursos de un sistema y compararlos con una base de datos de patrones de ataques previamente conocidos llamados firmas de ataques; no tienen la capacidad de detectar ataques nuevos.

Los sistemas basados en anomalías son usados en el ámbito de la investigación en gran parte debido a su alta tasa de falsos-positivos y falsos-negativos, pese a esto, guardan una ventaja sobre los sistemas basados en Misuse, la cual es su capacidad de encontrar posibles ataques a la red de datos que no sean previamente conocidos [3].

I-B. Base de datos KDD99

La base de datos KDD99 [4] ha sido y sigue siendo usada ampliamente por investigadores desde su creación en 1999

para el diseño y entrenamiento de los sistemas IDS; es una mejora de aquella creada en 1998 por los laboratorios Lincoln en el MIT, bajo el nombre de DARPA98.
La base de datos KDD99 consta del tráfico de red mediante un volcado TCPDUMP raw (binario) generado por siete semanas en una red militar, la cual fue procesada para formar alrededor de cinco millones de conexiones agrupadas en forma de vectores con 41 dimensiones cada uno, dónde cada dimensión representa una característica del tráfico de la red y cada vector es etiquetado cómo tráfico normal o algún tipo de ataque. Las primeras cinco semanas de evaluación fue generada la base de datos de entrenamiento (training dataset) del sistema IDS, la cual consta de 4898431 vectores, mientras que las últimas dos semanas fue generada la base de datos de prueba (test dataset), la cual consta de 311029 vectores.

I-B1. Tipos de ataques: cada uno de los ataques presentados tanto en el training dataset, cómo en el test dataset, pueden caer en alguna de las cuatro categorías siguientes, en [2] se puede encontrar una descripción detallada de cada uno de ellos; se describen de forma general de la siguiente manera:

- Denial of Service (DOS). Un atacante trata de impedir que un usuario legítimo pueda accesar a algún servicio. Ej. Syn Flooding
- Remote to local (R2L). Un usuario remoto trata de ganar acceso completo a alguna maquina de la red, ej. Password Guessing
- User to Root (U2R). Usuarios locales tratan de ganar privilegios de administrador, ej. El ataque Buffer Overflow
- Probe, Probing. Ataques que se basan en el escaneo de puertos, ej. Port Scanning.

I-B2. Problemática: El dataset KDD99 ha sido ampliamente utilizado para el entrenamiento y evaluación de sistemas IDS [5] [6] [7] por mencionar algunos, esto debido a que es de los pocos dataset disponibles para el entrenamiento de éstos sistemas que éste correctamente documentado desde su generación, atributos, instancias, y el tipo de ataques que contiene tanto el training dataset, cómo el test dataset. Sin embargo desde el año 2000 ha sido criticado por la generación sintética del tráfico que fue generado en la red militar, la sobrecarga del colector TCPdump al tratar de dumpear más información de la que puede manejar descartando una cantidad considerable de paquetes, y que no existe una definición exacta para cada uno de los ataques, permitiendo que sean interpretados en diferentes categorías por diferentes criterios [8]. De la misma manera en [9] se encontró evidencia de artefactos de simulación para la obtención del tráfico de la red militar. Por último en [10] se realiza un estudio exhaustivo del dataset KDD99 no en el contexto de la generación del tráfico en la red, sino en contexto de su contenido. En [10] se encontraron dos deficiencias graves en el dataset:

1. Conexiones redundantes: Existe un número enorme de conexiones redundantes, lo que causa que los algoritmos de entrenamiento de ML sean sesgados hacia los las conexiones más frecuentes (ataques DOS).
2. Nivel de Dificultad: No existe un dataset predefinido para el entrenamiento y prueba de los sistemas IDS, lo que provoca que algunos investigadores usen una porción del training dataset arbitraria, cómo training dataset y otra porción de ella cómo test dataset, provocando resultados de exactitud de al menos el 90% en los algoritmos de ML en la evaluación del IDS, sin modificar ningún parámetro adicional del algoritmo.

I-B. NSL- KDD Dataset
El NSL-KDD dataset es una solución propuesta en [10] a los problemas inherentes del KDD99 dataset para el entrenamiento de los sistemas IDS. NSL-KDD elimina las conexiones redundantes, mejora su distribución evitando el sesgado de los algoritmos de ML y reduce la cantidad de conexiones totales. La tabla 1 muestra la diferencia en la cantidad de instancias entre el training dataset KDD99 y NSL-KDD, mientras que la tabla 2 muestra la diferencia de la cantidad de instancias para el test dataset.

Tabla 1.- Training Dataset KDD99 vs NSL-KDD

Tipo de Ataque	KDD	NSL-KDD
Dos	3883370	45927
Probe	41102	11656
U2R	52	52
R2L	1126	995
Normal	972781	67343
Total	4898431	125973

De la tabla 1 se puede observar los cinco millones de conexiones que tiene almacenados el KDD99 Training Dataset a diferencia de los 125973 del NSL-KDD Training Dataset resultado de eliminar todos aquellos valores redundantes (duplicados).

Tabla 2.- Test Dataset KDD99 vs NSL-KDD

Tipo de Ataque	KDD	NSL-KDD
Dos	223298	5441
Probe	2377	1106
U2R	39	37
R2L	5993	2199
Normal	60593	9710
Ataques Nuevos	18730	3750
Total	311030	22543

De la misma manera la distribución de probabilidad de los tipos de ataques se encuentra menos concentrada en los ataques Dos y Normal. En la tabla 2 es posible observar una reducción considerable en la cantidad de instancias entre el KDD99 test dataset y NSL-KDD test dataset, además de la aparición de ataques nuevos que no se encuentran en el training dataset de ninguno de ellos, esto con el fin de evaluar

la capacidad de reconocer ataques nuevos, objetivo principal en los sistemas IDS.

II. Desarrollo

Muchos de los trabajos de investigación existentes para la evaluación de sistemas IDS se concentran en conseguir la mejor eficacia posible en el entrenamiento de los diferentes algoritmos de ML. En [11] se hace un estudio de la precisión en la clasificación de los diferentes algoritmos de ML, mostrando que la mayoría de ellos son capaces de conseguir una buena exactitud en su clasificación; no obstante no existe una comparación de la velocidad de entrenamiento y evaluación de los diferentes algoritmos de ML en los sistemas IDS para clasificar los tipos de ataques en el NSL-KDD dataset. Si bien no se ha hecho un estudio de la velocidad de entrenamiento de los sistemas IDS, dentro de un sistema IDS con fin práctico el rendimiento computacional es de suma importancia. Por lo antedicho, en éste trabajo se propone mejorar el tiempo de entrenamiento y evaluación de los sistemas IDS a través de la reducción de las 41 características encontradas en el NSL-KDD dataset, obteniendo aquellas que produzcan el menor tiempo de evaluación y entrenamiento, manteniendo a la par un alto resultado de exactitud en la clasificación de los algoritmos de ML. Los métodos para la selección de características se dividen esencialmente en dos categorías: métodos de tipo filtro y métodos de tipo envoltura [12]. Los métodos de envoltura usan el clasificador actual (algoritmo de ML) y su resultante probabilidad de error para seleccionar un subconjunto con menores características. El algoritmo de selección de características está ``envuelto'' dentro del propio clasificador. Los métodos del tipo filtro analizan las características independientemente del clasificador y usan una métrica para decidir que características se deben conservar [13]. Debido a los resultados de su clasificación cómo métrica, los algoritmos de tipo envoltura generalmente tienen un mejor desempeño que los algoritmos del tipo filtro.

Existen dos partes importantes en la selección de características, una es la estrategia de búsqueda y la otra el criterio de evaluación. Para la estrategia de búsqueda en [14] se encontró que los métodos de búsqueda heurística pueden ser ocupados de forma satisfactoria desempeñándose en problemas de optimización en datasets de gran escala, en consecuencia en éste trabajo se propone la selección de características en el NSL-KDD dataset a través un método de búsqueda heurística reciente llamado Ions Motion Optimization (IMO) [15]. IMO ha sido probado en tiempos de convergencia y calidad del resultado de optimización, en algoritmos de búsqueda heurística de trayectoria simple tales cómo el ascenso aleatorio de colinas con mutación aleatoria (RMHC) [16], el recocido simulado (SA) [17], además de métodos poblacionales cómo la optimización por enjambre de partículas (PSO) [18] y los algoritmos genéticos (GA) [19], ofreciendo mejores tiempos de convergencia incluso para el algoritmo RMHC, algoritmo cuyo principal fuerte son sus tiempos de convergencia. Para el criterio de evaluación se usa el algoritmo de ML SVMs debido a la velocidad de entrenamiento, factor importante en el diseño de sistemas IDS prácticos, demostrada en [13]; paralelamente de [13] se toma la función de rendimiento (fitness) para la estrategia de búsqueda del algoritmo IMO.

II-A. Selección de características usando IMO y SVM

El algoritmo IMO es uno de los miembros más recientes de las herramientas de optimización de búsqueda aleatoria. En general éstos algoritmos sufren de dos problemáticas encontradas, la taza de convergencia contra la calidad de la solución final. RMHC tiene altas tasas de convergencia, sin embargo es propenso a caer dentro de mínimos locales lo cual disminuye la calidad de su resultado, en contraste con GA el cual es capaz de salir de mínimos locales pero su tiempo de convergencia es bajo. IMO es capaz de salir mínimos locales gracias a su propiedad de fase líquida (exploración), además de ofrecer una alta tasa de convergencia gracias a su propiedad de fase sólida (explotación).

II-B. IMO

El algoritmo Ions Motion Optimzation (IMO), en español, optimización en base a movimiento de iones, toma su inspiración en base a los movimientos de atracción de los iones en la naturaleza. Es un algoritmo poblacional basado en el movimiento físico de los iones. IMO divide a la población de soluciones candidatas en partículas cargadas de forma positiva (cationes) y partículas cargadas de forma negativa (aniones). Se evalúan ambas poblaciones y se determina el mejor catión y anión de cada una; mediante iteraciones del algoritmo se empieza a aproximar la población de aniones hacía el mejor catión y paralelamente se empieza a aproximar la población de cationes hacia el mejor anión, a éste suceso dentro del algoritmo se le llama Fase líquida, encargada de la exploración del espacio de búsqueda, y simulando la facilidad que tienen los iones para moverse en un ambiente líquido. Una vez que las poblaciones han convergido hacia el mejor catión/anión, para evitar óptimos locales se expanden nuevamente los individuos de la población en el espacio de búsqueda alrededor del mejor catión/anión, a éste suceso dentro del algoritmo se le llama Fase sólida, la cual se encarga de la intensificación de las soluciones en el espacio de búsqueda. De forma general la figura 1 muestra el diagrama de flujo del algoritmo IMO con su fase sólida y líquida, una explicación más detallada del diagrama de flujo se puede encontrar en [15]

II-C. Aplicación de IMO y SVM

El diagrama de flujo de la figura 2 muestra el enfoque utilizado en éste trabajo para la reducción de las 41 características originales en el NSL-KDD dataset.

Al ser un algoritmo poblacional en vez de trayectoria simple cómo el mostrado en [13], es necesario crear una población de iones, separada en aniones P_a y cationes P_c dónde ambas poblaciones cuentan con un total de individuos $N/2$, los cuales poseen un total de 41 dimensiones cada uno $D = 41$ debido a las 41 características iniciales del NSL-KDD dataset y para su evaluación cada individuo de la población tiene su

propio rendimiento obtenido a través de la función fitness $F(S)$. Cada individuo en P_a y P_c es representado con una cadena binaria de 41 dimensiones S_i dónde i representa el indice del individuo. El 0 en S representa la ausencia de esa característica en el NSL-KDD dataset, mientras que el 1 representa su inclusión. Al iniciar la población, las cadenas S_i son generadas de forma aleatoria.

Es posible observar que a diferencia de la figura 1, en la figura 2 la función para evaluación del rendimiento (finess) $F(S)$ para el algoritmo IMO está dada únicamente por la probabilidad de error de clasificación obtenido mediante el algoritmo de ML SVMs puesto que en éste trabajo es utilizado el método de envoltura para la selección de características. Por lo tanto la función fitness para el algoritmo IMO está dada mediante la ecuación 1:

$$F(S) = P_{error}(S) = \frac{1}{C}\sum_{i=1}^{C}\gamma_i \qquad 1$$

Dónde $P_{error}(S)$ representa la probabilidad de error de clasificación del algoritmo SVMs al evaluar las características mencionadas de la cadena S en el NSL-KDD test dataset y siendo entrenado mediante el NSL-KDD training dataset. C representa el número de clase, y γ_i es la taza de error para la i-ésima clase.

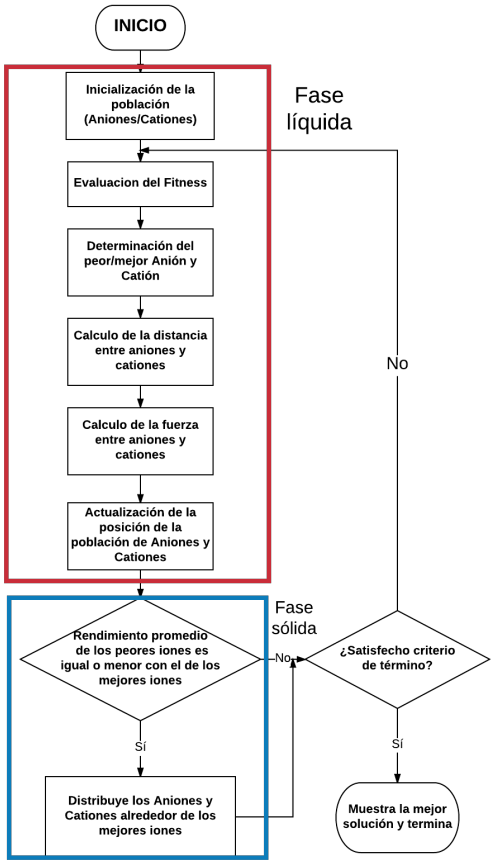


Figura 1.- Funcionamiento general del Algoritmo IMO

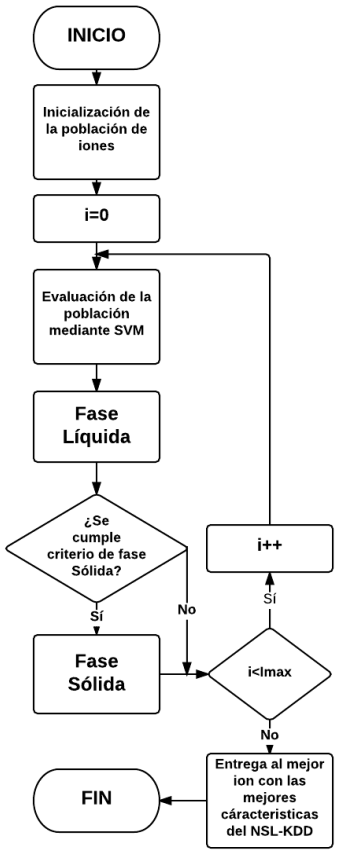


Figura 2- Algoritmo IMO y SVMs

II-D. Sistema IDS ligero basado en selección de características mediante IMO

El enfoque general de éste trabajo se muestra en la figura 2. Éste enfoque empieza generando la población de aniones P_a y cationes P_c. Generadas sus cadenas S_i se entrena el algoritmo SVMs a través del NSL-KDD training dataset. Una vez entrenado el algoritmo con las características seleccionadas en la cadena S_i se evalúan en el NSL-KDD test dataset obteniendo su probabilidad de error y por tanto su rendimiento (ecuación 1). Obteniendo el rendimiento se determinan los mejores individuos BP_a y BP_c de P_a y P_c respectivamente, comenzando la fase liquida con el fin de explorar el espacio de búsqueda, dónde dependiendo las diferencias de rendimiento se empiezan a aproximar en menor o mayor velocidad los demás individuos de la población hacia los mejores. Si el rendimiento promedio de los peores iones es igual o menor con el de los mejores iones, significa que los individuos de la población se encuentran muy cercanos en el espacio de búsqueda y el algoritmo entra en la fase sólida encargada de la explotación, separando a los individuos alrededor de los mejores, dicho de otra manera, la población de aniones se aproximará hacia el mejor catión y la población de cationes se aproximará hacia el mejor anión. Si la condición de parada se satisface, el algoritmo proporciona el mejor ion entre P_a y P_ca través del cual se extrae su cadena S_i y se entrena el sistema

IDS para ser evaluado. En éste trabajo se propone una población inicial de 10 individuos $N = 10$, y un total de iteraciones Máximas $I_{max} = 50$.

III. Resultados

Para el entrenamiento del sistema IDS se realizó un preprocesado de la base de datos NSL-KDD, a través de la normalización. La normalización es efectiva reduciendo el tiempo de aprendizaje del algoritmo SVMs. El objetivo de la normalización es tener una escala de medición común así cómo prevenir que haya preferencia sobre atributos con un alto rango, sobre otros con un rango menor [20]. La normalización se realizó a través del método de Min-Max, descrito mediante la ecuación 2.

$$NSL' = \frac{NSL_i - min_{NSL}}{max_{NSL} - min_{NSL}} \qquad 2$$

Dónde NSL' es el dataset normalizado, NSL es el dataset original, min_{NSL} es el mínimo valor encontrado para cada una de las características del dataset y max_{NSL} es el máximo valor encontrado para cada una de las características del dataset.

Dos experimentos fueron realizados para la clasificación del algoritmo SVMs del IDS en éste trabajo, el primero se realizó usando una clasificación binaria: 0 valor normal, 1 algún tipo de ataque. El segundo experimento se realizó ocupando las cinco clases principales para cada tipo de ataque: *Dos, Normal, Probe, U2R, R2L y Normal*.

El tiempo de evaluación para la convergencia del algoritmo IMO al reducir la cantidad de características del NSL-KDD dataset, así cómo las características finales seleccionadas para la clasificación binaria, y la clasificación con cinco clases se muestra en la tabla 3

Tabla 3.- Selección de características mediante IMO y SVMs

Clases	Tiempo (h)	Características	Total
2	5.45	3,4,5,6,11,14,25,29,30, 37	10
5	7.16	1,3,5,6,12,23,24,29,30, 33,34,35,36,38,40	15

De la tabla 3 se puede observar la disminución drástica de la cantidad de características del NSL-KDD dataset original, pasando de 41 características a 10 para el caso de la clasificación binaria, y a 15 características para la clasificación con 5 clases, es decir, una disminución de hasta el 75.6%.

Con las características mostradas en la tabla 3 se realiza el entrenamiento y evaluación del sistema IDS. La tabla 4 muestra el tiempo de entrenamiento y evaluación del sistema IDS para los dos tipos de clases así cómo con sus características totales (41) y reducidas (10 y 15). Es posible observar una drástica disminución del tiempo de entrenamiento para el caso de la clasificación binaria, yendo de un tiempo de $56.53\,s$ a $32.85\,s$ con únicamente 10 características. Paralelamente para el caso de 5 clases sus tiempos de entrenamiento se ven ampliamente reducidos, yendo de los $73.19\,s$ a los $35.95\,s$.

Tabla 4.- Tiempo Computacional del sistema IDS

Clases	Características	Entrenamiento (s)	Evaluación (s)
2	41	56.53	5.73
2	10	32.85	2.70
5	41	73.19	8.32
5	15	35.95	3.16

La tabla 5 muestra la matriz de confusión para el caso de la clasificación binaria y la tabla 6 muestra la matriz de confusión para el caso de la clasificación con cinco clases.

Tabla 5- Matriz de confusión del sistema IDS binario

41 Características			
-	**Normal**	**Ataque**	**% Aciertos**
Normal	9433	277	71%
Ataque	3855	8978	97%
Promedio			86%
10 Características			
-	**Normal**	**Ataque**	**% Aciertos**
Normal	9514	196	61%
Ataque	5996	6837	97%
Promedio			82%

Es claro que al existir una reducción de características inevitablemente habrá un reducción en su exactitud, sin embargo, el objetivo es mantener en lo posible la exactitud del sistema tratando a la vez de disminuir de forma considerable los tiempos de evaluación, ésto se puede observar en ambas tablas, dónde la exactitud promedio del sistema IDS disminuye del 86% al 82% para el caso de la clasificación binaria y del 78% al 75% para la clasificación en cinco clases. Ambos experimentos en la clasificación binaria y en cinco clases fueron realizados mediante un computador con las siguientes características: Sistema Operativo Mac OSX, Memoria RAM de 8 GB 1333 MHz DDR3 y un procesador a 2.3 GHz Intel Core i5.

IV. Conclusiones

Algunos trabajos han sido implementados usando la búsqueda heurística para la reducción de características enfocados en sistemas IDS basados en anomalías; éstos algoritmos han sido de trayectoria simple en su mayor parte y en menor cantidad del tipo poblacional. En éste trabajo fue aplicado de manera satisfactoria el algoritmo poblacional IMO junto con el algoritmo de ML SVMs. Los resultados de la tabla 3 en el tiempo de convergencia del algoritmo IMO si bien muestran unos tiempos de búsqueda altos, son menores que los mostrados en [13] pese a ser un algoritmo del tipo poblacional. Los tiempos altos en los algoritmos poblacionales son debido a la necesidad de entrenar al algoritmo SVMs con

cada individuo de la población, para después ser evaluado usando el NSL-KDD testing dataset. Contrariamente al tiempo de convergencia del algoritmo IMO, los beneficios después de obtener las mejores características se pueden apreciar en la tabla 4, reduciendo de forma notoria los tiempos de entrenamiento y evaluación del sistema IDS. Cómo se mencionó en [10] algunos trabajos muestran un desempeño en el sistema IDS de hasta 90% usando la base de datos KDD99 sin normalización en su preprocesado y sin ninguna selección de parámetros o modificación en su algoritmo de entrenamiento de ML, resultados prácticamente irreales. Éste trabajo posee un menor porcentaje de aciertos a comparación de los trabajos mostrados en [10] en la evaluación del IDS usando las 41 características debido al hacer uso del NSL-KDD dataset, y pese a tener un preprocesado en la base de datos. Aún con lo antedicho, se consigue una exactitud promedio del 82% para el caso de la clasificación binaria y del 75% haciendo uso de las cinco clases, con un tiempo de evaluación de apenas $2.70\,s$ y $3.16\,s$ respectivamente, tiempos lo suficientemente bajos para su aplicación en tiempo real.

Tabla 6- Matriz de confusión del sistema IDS con 5 clases

41 Características						
-	**Dos**	**Normal**	**Probe**	**R2L**	**U2R**	**Aciertos %**
Dos	6109	1074	163	112	0	96%
Normal	58	9395	248	7	2	68%
Probe	195	670	1556	0	0	76%
R2L	4	2520	17	213	0	64%
U2R	0	132	54	1	13	87%
Promedio						78%
10 Características						
-	**Dos**	**Normal**	**Probe**	**R2L**	**U2R**	**% Aciertos**
Dos	5270	2127	60	1	0	91%
Normal	38	9007	636	28	1	65%
Probe	429	465	1527	0	0	60%
R2L	5	2234	311	204	0	87%
U2R	50	126	22	2	0	0%
Promedio						75%

V. AGRADECIMIENTOS

Los autores agradecen al Instituto Politécnico Nacional, a la Sección de Estudios de Posgrados e Investigación de la Escuela Superior de Ingeniería Mecánica y Eléctrica unidad Zacatenco y al Consejo Nacional de Ciencia y Tecnología.

VI. REFERENCIAS

[1]Monowar H. Bhuyan, D. K. Bhattacharyya, y J. K. Kalita, Network Anomaly Detection: *Methods, Systems and Tools IEEE Communications Surveys Tutorials*, Vol. 16, No 1. First Quarter. 2014

[2]H. Gunes, Kayacik, A. Nur Zincir-Heywood y Malcolm I. Heywood, Selecting Features for Intrusion Detection: A Feature Relevance Analysis on KDD 99 *Intrusion Detection Dataset Dalhousie University*, Faculty of Computer Science

[3]G. Di Crescenzo. A. Ghosh y R. Talpade. Towards a theory of Intrusion Detection. *Lecture Notes in Computer Science.* Vol 39. 2005

[4]Base de datos KDD99 disponible en, http://www.kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, Septiembre. 2017.

[5]P. Ganesh Kumar y D. Devaraj, Network Intrusion Detection Using Hybrid Neural Networks, *IEEE-ICSCN* 2007.

[6]Dilip Kumar Barman y Dr. Guruparasad Khataniar, Design of Intrusion Detection System Based on Artificial Neural Network and Application of Rough Set, *International Jornal of Computer Science of Communication Networks,* 2011

[7]Maheskumar Sabhanani y Gursel Serpen Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset within Misuse Detection *Context EECS University of Toledo*, 2002

[8]J.McHugh Testing intrusion detection systems: a critique of the 1998 and 199 DARPA intrusion detection system evaluations as performed by lincoln laboratory ACM. *Transactions on Information and System Security*, vol 3. No 4. 2000.

[9]M.Mahoney y P.Chan, An Analysis of the 1999 DARPA/Lilcoln Laboratory Evaluation Data for Network Anomaly Detection *Lecture Notes in computer Science*, 2003

[10]Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu y Ali A. Ghorbani. A detailed Analysis of the KDD CUP 99 Data Set *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications*. 2009

[11]Nigel Williams, Sebastian Zander y Grenville Armitrage, A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *ACM SIGCOMM Computer Communication Review*, Volumen 36 5- 13, 2006

[12]You Chen, Yang Li, Xue Qi Cheng y Li Guo. Survey and Taxonomy of Feature Selection Algoritms in Intrusion Detection System. *Conference on Information Security and Criptology*. LNCS4318, 153-167,2006

[13] YouChen, Wen-Fa Li y Xue-Qi Cheng, Toward Building Lightweigh Intrusion Detection System Through Modified RMHC and SVM, *Institute of Computing Technology, Chinese Academy of Sciences.*

[14]A.K. Jain y D. Zongker, Feature selection: Evaluation, application, and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no.2, 153-158, 1997.

[15]Behzad Javidy, Abdolreza Hatamlou y Seyedali Mirjalili, Ions Motion Algorithm for solving optimization problems

Departament of Computer Science, Islamic Azad University, 2015

[16]Stephanie Forrest y Melanie Mitchel, Relative Building-Block Fitness and the Building-Block Hypothesis *Whitley Foundations of Genetic Algorithms 2*, San Mateo, CA, 1993

[17]Emile Aarts, Jan Kors y Wil Michiels, Simulated Annealing, *Search Metodologies*, Chapter 7, pg 188-210

[18]James Kennedy y Ruseell Eberhart, Particle Swarm Optimization *Purdue School of Engineering and Tecnology*, 1995

[19]Kumara Sastry y David Goldberg, Genetic Algorithms, *Search Metodologies*, Chapter 4, pg 98-125

[20]Luai A. Shlabi, Zyad Shaaban y Base Kassabeh, *Data Mining a Preprocessing Engine Computer Sci*ence. pg 735-739. 2009

Dejaré ésta rosa en el abandono,
el abandono está lleno de rosas.

Saturación
Mario Benedetti