Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

Clasificación de Ataques a una Red de Telecomunicación con Deep Learning

Autora: María Dolores Trapero Estepa

Tutor: Juan José Murillo Fuentes

Francisco Javier Payán Somet

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Trabajo Fin de Máster
Máster Universitario en Ingeniería de Telecomunicación

Clasificación de Ataques a una Red de Telecomunicación con Deep Learning

Autora:
María Dolores Trapero Estepa

Tutor:
Juan José Murillo Fuentes
Catedrático de Universidad
Francisco Javier Payán Somet
Profesor titular de Universidad

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo Fin de Máster: Clasificación de Ataques a una Red de Telecomunicación con Deep Learning

Autora: María Dolores Trapero Estepa

Tutor: Juan José Murillo Fuentes

Francisco Javier Payán Somet

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

Agradecimientos

Dos años y medio después de exponer mi Trabajo Fin de Grado, otra vez me encuentro escribiendo estas líneas en último lugar, aunque paradójicamente sean las que dan comienzo al desarrollo de este documento. De nuevo vuelvo a poner el broche final a otro reto, el Máster en Ingeniería de Telecomunicación. Han sido dos años muy duros y de mucho trabajo, pero que me han curtido y me han enseñado aún más. Como epílogo quedan estas páginas, después de 9 meses de aprendizaje y buceo por la inteligencia artificial que me han servido para confirmar algo que ya intuía: que es un tema tan amplio como realmente apasionante.

Haciendo honor al título que corona esta hoja, primeramente, agradezco a mi tutor Juan José Murillo la libertad y autonomía concedida desde un primer momento, indicándome los aspectos que él consideraba convenientes para que el trabajo fuese lo mejor posible pero siempre dejándome la batuta de todo esto.

A mi familia, en especial a mis padres y mi hermana; ellos son los que durante más horas me han visto dedicada en cuerpo y alma a este trabajo y los que más me han arropado. A mi segunda familia, mis amigos: mi agradecimiento infinito por tener siempre las palabras de aliento necesarias para ayudarme a seguir adelante. Y a los que por la inercia de la vida ya no puedo contarles en persona que por fin soy ingeniera de pleno derecho: sé que estaríais muy orgullosos de lo que he conseguido.

Gracias a todas y cada una de estas personas, por hacerme feliz y por hacerme creer que soy capaz de todo.

María Dolores Trapero Estepa

Sevilla, 2021

Resumen

Los sistemas de seguridad de la información en las organizaciones requieren de mecanismos de protección complejos para evitar poner en compromiso sus datos. El hecho de que en la actualidad tanto usuarios internos como externos a la red corporativa de la empresa puedan conectarse de forma local o remota aumenta las posibilidades de recibir un ataque. Para defender a la organización de este tipo de accesos se han desarrollado (entre otros) los sistemas de detección de intrusos (IDS) que identifican tráfico malicioso en la red, y los sistemas de prevención de intrusos (IPS), que una vez han detectado y aprendido el ataque, lo bloquean e incluso pueden contrarrestarlo.

El objetivo del presente Trabajo Fin de Máster es el diseño y posterior evaluación de un prototipo de clasificador susceptible de ser implementado como sistema de detección de intrusos en redes (NIDS) utilizando un autocodificador variacional, técnica de aprendizaje profundo empleada en estudios recientes sobre el tema. Concretamente, se procederá a la evaluación del desempeño del autocodificador variacional como clasificador binario y multiclase a través de la comparativa con otros algoritmos de aprendizaje máquina.

Abstract

Information security systems in organizations require complex protection mechanisms to avoid compromising their data. The fact that nowadays both internal and external users to the company's corporate network can connect locally or remotely increases the chances of receiving an attack. To defend the organization from this type of access, intrusion detection systems (IDS) have been developed (among others), which identify malicious traffic on the network, and intrusion prevention systems (IPS), which, once they have detected and learned the attack, block it and can even counter it.

The objective of this Master Thesis is the design and subsequent evaluation of a prototype classifier that can be implemented as a network intrusion detection system (NIDS) using a variational autoencoder, a deep learning technique used in recent studies on the subject. Specifically, the performance of the variational autoencoder as a binary and multiclass classifier will be evaluated by comparing it with other machine learning algorithms.

Introducción

Desde el nacimiento del primer ordenador personal en 1981 a manos de IBM hasta nuestros días, la sociedad ha avanzado de manera irrefrenable hacia un mundo cada vez más digital en el que la tecnología se convierte en parte imprescindible de la vida de un ser humano: desde recurrir a ella como mero entretenimiento hasta ser un elemento esencial en el día a día para realizar cualquier tipo de acción: trabajar, consultar movimientos bancarios, realizar un trámite burocrático, comprar ropa... La inmediatez y la comodidad (entre otros beneficios) que supone realizar este tipo de actos sin tener que moverse de casa resulta muy positiva; en contraposición, obliga a introducir datos sensibles en páginas web, que pueden acabar en manos equivocadas si no se toman las medidas de seguridad pertinentes para preservar la información privada. Es precisamente esto último el principal propósito de la ciberseguridad: proteger los sistemas contra ataques informáticos que pretendan hacerse con el control de los dispositivos o robar datos.

La ciberseguridad resulta una preocupación importante a nivel global; de hecho, el Foro Económico Mundial colocó los ciberataques como una de las principales amenazas mundiales en términos de impacto en su mapa de riesgos globales de 2020 [1]. Según un estudio realizado por Microsoft y la evaluadora de riesgos Marsh en materia de ciberriesgo, las empresas cada vez reconocen más la importancia de proteger su red, pero no invierten lo suficiente en tecnología de ciberseguridad. Aunque el mercado de la ciberseguridad movió más de 124 millones de dólares en 2019, se estima que globalmente, el coste que suponen los ciberataques a las organizaciones asciende al trillón de dólares [2].

Pero no sólo son los dispositivos informáticos tradicionales (como servidores y portátiles) las únicas víctimas de ciberataques. El internet de las cosas cada vez juega un papel más relevante en distintas áreas como movilidad, salud, logística etc., haciendo que el número de dispositivos con conectividad a la red crezca año tras año y, por ende, la probabilidad de encontrar vulnerabilidades en ellos. Según se menciona en un estudio realizado en 2015 por la consultoría EY, se estimó que aproximadamente el 70% de los dispositivos IoT presentan vulnerabilidades [3]. Precisamente aprovechando estos agujeros de seguridad en dispositivos IoT el malware Silex, creado en 2019 por un adolescente, consiguió inutilizar más de 2.000 dispositivos IoT (cámaras, bombillas, routers…) en tan sólo 4 horas [4]. Y este es solamente uno de los muchos *botnets* que tienen como objetivo dispositivos IoT.

Centrándonos más en el ámbito de la seguridad de la información en las organizaciones, éstas requieren de sistemas de protección complejos para evitar poner en compromiso sus datos. El hecho de que en la actualidad tanto usuarios internos como externos a la red corporativa de la empresa puedan conectarse de forma local o remota aumenta las posibilidades de recibir un ataque. Esta razón ha sido el principal acicate para desarrollar diversas herramientas hardware y software para detectar y prevenir accesos maliciosos a la red [5]. Para evitar ataques externos se suelen utilizar cortafuegos (que restringen el tráfico de servicios desconocidos bloqueando puertos) y redes privadas virtuales. Otro tipo de soluciones contra los ciberataques pasan por segmentar la red asociando diferentes niveles de seguridad, o bien utilizar metodologías de prevención de pérdida de datos que protegen al personal (en muchas ocasiones, el eslabón más débil del sistema de seguridad) de poner en peligro datos sensibles [6]. La dificultad se presenta cuando el ataque se incluye en un paquete cuyo tráfico de servicio está reconocido por la red, y la ofensiva se despliega desde dentro de la red corporativa.

Para defender a la organización de este tipo de accesos se han desarrollado sistemas de detección de intrusos (IDS) que identifican tráfico malicioso en la red, y los sistemas de prevención de intrusos (IPS), que una vez han detectado y aprendido el ataque, lo bloquean e incluso pueden contrarrestarlo. Ambos sistemas pueden aplicar dos tipos de metodología a la hora de detectar los ciberataques: o bien con un método basado en firmas (comparando los ataques con una base de datos de firmas) o con un método basado en anomalías (empleando un algoritmo de aprendizaje). El objetivo de los investigadores centrados en este campo se centra en la búsqueda de algoritmos que ofrezcan unos niveles de precisión en la detección muy altos, para poder ser de esta forma implementados en sistemas de detección de intrusos.

Una de las vías de investigación principales a fin de crear algoritmos de detección de ataques que funcionen adecuadamente para poder ser usados como IDS consiste en implementar estos métodos usando inteligencia artificial (IA). Concretamente, se destaca el desarrollo de técnicas de *machine learning* para generar sistemas de seguridad flexibles y que sean capaces de identificar y adaptarse a los cambios del malware más reciente. Como un subconjunto del *machine learning* encontramos el *deep learning* o aprendizaje profundo, en el que se predica con la idea de aprendizaje desde el ejemplo, haciendo que el propio modelo aprenda automáticamente y se mejore paulatinamente su inteligencia para que pueda hacer frente a nuevos ataques de forma independiente [7]. El abanico de algoritmos de aprendizaje máquina utilizados en los estudios actuales resulta variado; se pueden encontrar métodos como el KNN, SVM o árboles de decisión, entre otros. Lo mismo ocurre con las técnicas de preprocesado y, en especial, de selección de características para las bases de datos. En lo referente a los métodos propuestos de *deep learning* en materia de sistemas de detección de intrusos, existen algoritmos como los autocodificadores (variacionales, dispersos…), las *deep belief networks* (DBN) o las *convolutional neural networks* (CNN), entre otros. [8], [9].

Existen muchos factores a tener en cuenta a la hora de acometer el diseño de un prototipo de IDS teniendo como base el aprendizaje máquina: la selección de una base de datos etiquetada con las características adecuadas que permita entrenar y testar el modelo, decidir si se va a efectuar una clasificación binaria o multiclase, la elección del preprocesado adecuado para dicha base de datos, la posterior elección del tipo de algoritmo de clasificación que se usará, las métricas utilizadas para evaluar el rendimiento… Los inconvenientes que pueden surgir a lo largo del proceso de diseño son numerosos, sobre todo en el preprocesado, y algunos pasos resultan críticos para el devenir del modelo. Uno de estos pasos es la selección de las características adecuadas que mejor clasifican las entradas de una base de datos. Esta etapa es muy importante cuando se dispone de un set extenso con numerosas características, ya que el tiempo de ejecución de un algoritmo de clasificación con gran número de atributos puede llegar a ser bastante elevado. El objetivo de ejecutar este paso es el de llegar a una solución de compromiso en tiempo de respuesta computacional y buenos resultados de clasificación. Esta técnica aparecerá aplicada en el desarrollo de uno de los códigos de este documento.

El objetivo del presente Trabajo Fin de Máster es el diseño y posterior evaluación de un prototipo de clasificador susceptible de ser implementado como sistema de detección de intrusos en redes (NIDS) utilizando un autocodificador variacional, técnica de aprendizaje profundo empleada en estudios recientes sobre el tema. Profundizando más en el contenido de este texto, se procederá a la evaluación del desempeño del autocodificador variacional como clasificador binario y multiclase a través de la comparativa con otros algoritmos de aprendizaje máquina. Los modelos se entrenarán y evaluarán con la misma base de datos de ataques a una red de comunicaciones, la cual será previamente preprocesada y estudiada para visualizar sus particularidades. Una vez se hayan extraído las métricas correspondientes a la clasificación binaria y multiclase de todos los modelos, se deducirán una serie de conclusiones sobre el desempeño del autocodificador variacional en tareas de clasificación de ataques.

El contenido de este texto se estructura de la siguiente forma: los primeros capítulos tienen un corte teórico para introducir al lector en materia, comenzando con unas nociones acerca de la ciberseguridad y los sistemas de detección de intrusos, y prosiguiendo con un capítulo centrado en la inteligencia artificial: definición de terminología, explicación de clasificadores y flujo de trabajo del diseño de un algoritmo de *machine learning*. El tercer capítulo recoge una recopilación de bases de datos de ataques a una red de telecomunicaciones.

Los Capítulos 4 y 5 recogen el desarrollo práctico del trabajo. En el Capítulo 4 se estudia la composición de la base de datos escogida para entrenar los algoritmos, y se desarrollan dos modelos de árboles de decisión como clasificadores binario y multiclase. Mientras tanto, en el Capítulo 5 se desarrollan y entrenan tanto el autocodificador variacional como un perceptrón multicapa, y se compararán globalmente los resultados en clasificación binaria y multiclase del VAE con respecto a los otros dos algoritmos desarrollados.

En último lugar, en el Capítulo 6 se explicará cómo poner en explotación un sistema de estas características con la finalidad de ser utilizado por una empresa; mientras que en el Capítulo 7 se enumerarán las conclusiones derivadas de los resultados, así como posibles vías de continuidad del trabajo aquí desarrollado, y se tratarán las posibles vías de investigación que se abren para los sistemas de detección de intrusos teniendo en cuenta los estudios recientes sobre este tema.

Índice

Agradecimientos **7**

Resumen **9**

Abstract **11**

Introducción **13**

Índice **17**

Índice de Tablas **19**

Índice de Figuras **21**

1 Fundamentos teóricos sobre ciberseguridad **25**
1.1 ¿Qué es la ciberseguridad? *25*
1.1.1 Arquitectura básica del sistema de seguridad de una red corporativa 27
1.2 Clasificación de ataques a una red *28*
1.2.1 Ataques basados en la web 28
1.2.2 Ataques basados en el sistema 33
1.2.3 Otra clasificación de ataques 34
1.3 Sistemas de detección de intrusos (IDS) y sistemas de prevención de intrusos (IPS) *35*
1.3.1 Sistemas de detección de intrusos (IDS) 36
1.3.2 Sistemas de prevención de intrusos (IPS) 37

2 Inteligencia artificial: un nuevo impulso a la ciberseguridad **39**
2.1 Inteligencia artificial, machine learning y deep learning *39*
2.2 Clasificaciones de algoritmos de ML *41*
2.2.1 Aprendizaje supervisado, no supervisado, semisupervisado y por refuerzo 42
2.2.2 Aprendizaje online y aprendizaje por lotes 44
2.2.3 Aprendizaje basado en instancias y aprendizaje basado en el modelo 44
2.3 Algoritmos de clasificación *45*
2.3.1 Naïve Bayes 45
2.3.2 k-Nearest neighbors (KNN) 45
2.3.3 Máquinas de soporte vectorial (SVM) 46
2.3.4 Árboles de decisión 48
2.3.5 Red neuronal profunda (DNN) 51
2.3.6 Red neuronal convolucional (CNN) 53
2.3.7 Red neuronal recurrente (RNN) 55
2.3.8 Máquina de Boltzmann restringida (RBM) 55
2.3.9 Autocodificadores 56
2.4 Flujo de trabajo del diseño de un algoritmo de ML *57*
2.4.1 Obtención de datos y visualización de su estructura 57

2.4.2 Preprocesado de datos 59
2.4.3 Elección del modelo, entrenamiento y evaluación 61
2.4.4 Configuración de hiperparámetros 61
2.4.5 Lanzamiento y monitorización del sistema 61

3 Bases de datos relativas a la detección de intrusos 65
3.1 KDD'99 (1998-99) 65
3.2 NSL-KDD (2009) 67
3.3 UNSW-NB15 (2015) 68
3.4 Aegean Wi-Fi Intrusion Dataset (AWID) (2016) 69
3.5 CICIDS2017 (2017) 70
3.6 Otras bases de datos existentes 72

4 Random forest como clasificador para sistemas de detección de intrusos 75
4.1 Recopilación de datos y visualización de la base de datos 75
4.1.1 UNSW-NB15 76
4.2 Preprocesado de los datos 83
4.2.1 Artículo I [67] 84
4.2.2 Artículo II [77] 86
4.3 Elección del modelo, entrenamiento y evaluación 87
4.3.1 Artículo I 88
4.3.2 Artículo II 89
4.4 Ajuste de hiperparámetros 92
4.4.1 Artículo I 93
4.4.2 Artículo II 93
4.5 Mejores resultados obtenidos y comparación con resultados de los artículos 94
4.5.1 Artículo I 94
4.5.2 Artículo II 94

5 Deep learning para sistemas de detección de intrusos 97
5.1 Deep learning y ciberseguridad 97
5.2 Autocodificador variacional (VAE) 99
5.2.1 Estructura y funcionamiento de un VAE 99
5.2.2 Fundamentos matemáticos de un VAE 101
5.3 Desarrollo de un CVAE para un sistema de detección de intrusos 108
5.3.1 Autocodificador variacional condicional (CVAE) 108
5.3.2 Procedimiento para utilizar el CVAE como clasificador 110
5.3.3 Desarrollo del modelo y resultados 111
5.4 Desarrollo y evaluación de un MLP 118
5.5 Comparación de resultados de CVAE con el resto de modelos y conclusiones 121

6 Explotación de un IDS en la realidad 125
6.1 Análisis de requisitos y diseño 125
6.1.1 Modelo de funcionamiento de un sistema de detección de intrusos 126
6.1.2 Elección de elementos del sistema completo 129
6.2 Implementación del sistema completo 136

7 Conclusiones y líneas de futuro 139
7.1 Conclusiones globales y líneas futuras relativas a la optimización del trabajo 139
7.2 Líneas futuras relativas al desarrollo de los algoritmos para sistemas de detección de intrusos 140

8 Bibliografía 143

Anexo A: Herramientas utilizadas en el desarrollo de los algoritmos 149

Bibliografía de los anexos 155

Anexo B: Códigos usados en el trabajo 157

ÍNDICE DE TABLAS

Tabla 1. Archivos del dataset NSL-KDD. 68

Tabla 2. Features del tráfico del dataset UNSW-NB15. En naranja aparecen los atributos de flujo, en amarillo, los atributos básicos, en azul, los atributos de contenido; en verde, las características relativas al tiempo, en rosa, características adicionales; en verde agua se representan las caraterísticas de etiqueta. 76

ÍNDICE DE FIGURAS

Figura 1: Ejemplo de una red segura que utiliza IDS/IPS, una red DMZ y un firewall. Fuente: SecureWorks 27

Figura 2: Esquema sencillo de realización de un ataque de inyección SQL. Fuente: Universidad Rey Juan Carlos 29

Figura 3: Esquema sencillo de un ataque Man in the Middle. Fuente: tecnonucleous.com 30

Figura 4: Ejemplo de phishing, intentando replicar un correo de PayPal. Fuente: malwarebytes.com 32

Figura 5: Clasificación de los IDS atendiendo a distintas categorías. 36

Figura 6: Organización jerárquica de la inteligencia artificial, el aprendizaje máquina y el aprendizaje profundo. Fuente: medium.com 40

Figura 7: Esquema básico que ejemplifica la tarea de clasificación de un algoritmo de ML. Fuente: medium.com 42

Figura 8: Ejemplo de clasificación del algoritmo KNN. Si K=3, el cuadrado amarillo pertenecerá a la clase B. Si K=7, el cuadrado amarillo pertenecerá a la clase A. Fuente: datacamp.com 46

Figura 9: Ejemplo de la separación de clases realizada por una SVM. Fuente: medium.com 47

Figura 10: Ejemplo de utilización de una función kernel para separar las clases existentes. Fuente: medium.com 47

Figura 11: Estructura de un árbol de decisión. Fuente: javatpoint.com 48

Figura 12: Ejemplo para calcular el valor chi-square. Fuente: analyticsvidhya.com 50

Figura 13: Estructura de un perceptrón. Fuente: deepAI.com 52

Figura 14: Estructura simplificada de una red neuronal profunda. Cada círculo representa una neurona. Fuente: A Review of Deep Learning Security and Privacy Defensive Techniques. Tariq, I. (2020) 52

Figura 15: Estructura de una CNN. Fuente: [42] 53

Figura 16: Operación de convolución realizada en las capas de convolución. I es la imagen que se introduce como entrada en la red, K es el kernel e I*K es la salida de la capa convolucional, resultado de efectuar la operación de convolución. 54

Figura 17: Diferencia entre una celda de una RNN (figura de arriba) y una celda LSTM (figura de abajo). Fuente: O'Reilly 55

Figura 18: Estructuras de una máquina de Boltzmann genérica y una máquina de Boltzmann restringida. Fuente: Real-Time Classification and Sensor Fusion with a Spiking Deep Belief Network. O'Connor, P. et al. (2013) 56

Figura 19: Estructura de una matriz de confusión para una clasificación multiclase. Fuente: "Activity, Context, and Plan Recognition with Computational Causal Behaviour Mode" – Frank Krüger 62

Figura 20: Ejemplo de curva ROC. Fuente: towardsdatascience.com 64

Figura 21: Mapa de posición de los elementos en el entorno SOHO. 70

Figura 22: Comparación entre la CICIDS2017 y otras bases de datos atendiendo a las 11 características críticas para realizar una BBDD de detección de intrusos. 71

Figura 23: Recuento de instancias por categoría en el set de datos completo. 79

Figura 24: Recuento de instancias por categoría en el set de datos reducido. 79

Figura 25: Matriz de correlación para el set de datos completo. 80

Figura 26: Matriz de correlación para el set de datos reducido. 81

Figura 27: Resultado de aplicar t-SNE al conjunto de datos completo. 82

Figura 28: Resultado de aplicar t-SNE al conjunto de datos reducido. 83

Figura 29: Entorno gráfico de WEKA. Después de cargar el .CSV del training set, se marcan las características id y attack_cat y se eliminan con "Remove". En "Choose" elegimos la función Resample para dividir el dataset en entrenamiento y test. 84

Figura 30: Submenú que aparece al clicar en la barra blanca de "Attribute Evaluator". Se muestran distintos parámetros de personalización de la evaluación. En la pestaña "Classifier" se escoge el tipo de modelo que realizará la evaluación de las características. 85

Figura 31: Parte del código común a articulo_I.py y a articulo_II.py que se encarga del entrenamiento del clasificador y posterior predicción con el set de test. La función performance extrae las métricas de interés para evaluar el rendimiento del clasificador. 87

Figura 32: Resultados obtenidos de la evaluación del clasificador del Artículo I. El valor "0" se corresponde con tráfico normal, mientras que el valor "1" se corresponde con tráfico anómalo (ataques). El campo "support" muestra el número de instancias existentes de cada clase. 88

Figura 33: Matriz de confusión correspondiente a los resultados del Artículo I. 89

Figura 34: Resultados obtenidos de la evaluación del clasificador binario del Artículo II. El valor "0" se corresponde con tráfico normal, mientras que el valor "1" se corresponde con tráfico anómalo (ataques). El campo "support" muestra el número de instancias existentes de cada clase. 90

Figura 35: Matriz de confusión correspondiente a los resultados del clasificador binario del Artículo II. 90

Figura 36: Resultados obtenidos de la evaluación del clasificador multiclase del Artículo II. 91

Figura 37: Matriz de confusión correspondiente a los resultados del clasificador multiclase del Artículo II. 92

Figura 38: Rejilla para realizar el ajuste de hiperparámetros. Común a articulo_I.py y articulo_II.py 92

Figura 39: Resultado de la ejecución de la búsqueda en rejilla para optimizar la selección de hiperparámetros para el random forest del Artículo I. 93

Figura 40: Comparación entre los resultados obtenidos en el presente trabajo con los expuestos en el Artículo I. 94

Figura 41: Comparación entre los resultados obtenidos en el presente trabajo con los expuestos en el Artículo II para el caso de clasificación binaria. 95

Figura 42: Comparación entre los resultados obtenidos en el presente trabajo con los expuestos en el Artículo II para el caso de clasificación multiclase. 96

Figura 43: Gráfico que ilustra las distintas tendencias a lo largo del tiempo acerca de las soluciones empleadas en aplicaciones de ciberseguridad. Fuente: informationage.com 98

Figura 44: Estructura de un autocodificador variacional. Fuente:towardsdatascience.com 100

Figura 45: Modelo gráfico del problema. Las líneas continuas denotan el modelo generativo $p_\theta(z)p_\theta(x|z)$, las líneas discontinuas denotan la aproximación $q_\phi(x|z)$ a la probabilidad intratable $p_\theta(x|z)$. Los parámetros variacionales ϕ se aprenden de forma conjunta con los parámetros generativos θ del modelo. Fuente: renom.jp 101

Figura 46: Representación gráfica del procedimiento a seguir por un algoritmo de descenso de gradientes. Fuente: mc.ai 105

Figura 47: Ilustración de la técnica de reparametrización. Se desplaza la aleatoriedad existente en z al reparametrizar esta variable como una función determinista y diferenciable respecto de ϕ, x, y una nueva variable ϵ. Esto permite la propagación hacia atrás y el cómputo de los gradientes. 107

Figura 48: Comparación de la estructura del CVAE con la arquitectura del VAE. Fuente: [100]. 109

Figura 49: Estructura para emplear el CVAE como clasificador [100]. 110

Figura 50: Comparación de los resultados obtenidos para la NSL-KDD entre el artículo [100] y el código de GitHub. Se ha seleccionado “weighted avg” en las métricas del código de GitHub para realizar la comparativa (es la que se usa en el artículo). 111

Figura 51: Estructura de capas del codificador y decodificador para el CVAE, adaptado a la UNSW-NB15. Los números en azul se corresponden con el tamaño de la entrada de la capa o longitud del vector de etiquetas de clase L para una clasificación binaria. Los números en rojo se corresponden con el tamaño de la entrada de la capa o longitud del vector de etiquetas de clase L para una clasificación multiclase. Figura adaptada de [100]. 113

Figura 52: Gráfica que representa la función de pérdida frente a los epochs para un entrenamiento binario. 114

Figura 53: Gráfica que representa la función de pérdida frente a los epochs para un entrenamiento multiclase. 115

Figura 54: Resultados obtenidos por el CVAE para una clasificación binaria. “El valor “0” se corresponde con tráfico normal, mientras que el valor “1” se corresponde con tráfico anómalo (ataques). El campo “support” muestra el número de instancias existentes de cada clase. 115

Figura 55: Matriz de confusión correspondiente a los resultados del CVAE como clasificador binario. 116

Figura 56: Resultados obtenidos por el CVAE para una clasificación multiclase. “support” indica el número de muestras del conjunto de test pertenecientes a cada clase. 116

Figura 57: Matriz de confusión correspondiente a los resultados del CVAE como clasificador multiclase. 117

Figura 58: Mejor configuración de hiperparámetros del MLP para clasificación binaria y multiclase. La función de activación óptima (parámetro act) es una sigmoide en clasificación binaria, y ReLu en clasificación multiclase). 118

Figura 59: Resultados obtenidos por el MLP para una clasificación binaria. El valor “0” se corresponde con tráfico normal, mientras que el valor “1” se corresponde con tráfico anómalo (ataques). El campo “support” muestra el número de instancias existentes de cada clase. 118

Figura 60: Matriz de confusión correspondiente a los resultados del MLP como clasificador binario. 119

Figura 61: Resultados obtenidos por el MLP para una clasificación multiclase. “support” indica el número de muestras del conjunto de test pertenecientes a cada clase. 119

Figura 62: Matriz de confusión correspondiente a los resultados del MLP como clasificador multiclase. 120

Figura 63: Comparativa global de resultados entre todos los modelos para una clasificación binaria. 122

Figura 64: Comparativa global de resultados entre todos los modelos para una clasificación multiclase. 122

Figura 65: Comparativa global de resultados según clases tomando como referencia el F1-Score. 123

Figura 66: Modelo general de gestión de seguridad. Fuente: [102] 126

Figura 67: Diagrama de alto nivel del camino que sigue el flujo de tráfico cuando existe un IDS en la red. 130

Figura 68: Ejemplo de diagrama de red de una empresa pequeña. 132

1 FUNDAMENTOS TEÓRICOS SOBRE CIBERSEGURIDAD

"It takes 20 years to build a reputation and few minutes of cyber-incident to ruin it."

-Stéphane Nappo-

La seguridad informática o ciberseguridad es un tema presente desde el primer momento en que los ordenadores comenzaron a interconectarse entre sí a través de redes de comunicaciones. No obstante, el enorme crecimiento de Internet, unido a otros sucesos como la explosión del Internet de las Cosas (IoT) o las ciudades inteligentes, han propiciado que este tema tome un cariz bastante relevante, debido a que el progreso en la materia obliga a que los sistemas de seguridad informática se vuelvan cada vez más y más complejos.

En este capítulo se propone una revisión de términos relacionados con la ciberseguridad, enfocándose sobre todo en aquellos que serán tratados más en profundidad en el presente Trabajo Fin de Máster como, por ejemplo, los sistemas de detección de intrusos en una red o la tipología de ataques existentes hoy día.

1.1 ¿Qué es la ciberseguridad?

La ciberseguridad es el área de las ciencias de la computación encargada del desarrollo y la implementación de mecanismos de protección de la información y de la infraestructura tecnológica [10]. Este campo de estudio cobra especial importancia en el mundo interconectado en el que vivimos actualmente; un ciberataque puede resultar en el robo de datos sensibles u ocasionar la caída de equipos críticos, de ahí la obligación de mantener segura cualquier tipo de infraestructura.

La ciberseguridad se puede aplicar en multitud de contextos, desde dispositivos móviles a seguridad aplicada a negocios, e incluso se puede subdividir en las siguientes categorías [11]:

- **Seguridad en redes:** Es la práctica de proteger una red informática de los intrusos que quieran acceder a ella, ya sea de atacantes que quieren comprometer una entidad específica de la red, o de malware.

- **Seguridad de aplicación:** Se centra en asegurar que las aplicaciones creadas están a prueba de amenazas y que no poseen agujeros de seguridad. Una aplicación vulnerable puede poner en compromiso datos sensibles recogidos por la propia aplicación. Para garantizar el éxito en lo que a seguridad se refiere, la etapa de diseño de la aplicación es crítica.
- **Seguridad de la información:** La seguridad de la información protege la privacidad e integridad de los datos, tanto los que se hallan almacenados como los que viajan desde un origen a un destino.
- **Seguridad operativa:** La seguridad operativa incluye acciones tales como la asignación de permisos a un usuario al acceder a una determinada red, y los procedimientos que determinan cómo y dónde se pueden almacenar o compartir los datos.
- **Recuperación de desastres y continuidad del negocio:** Esta categoría se centra en dar respuesta a cómo una organización o empresa responde a una brecha de seguridad o cualquier otro evento que ocasione la pérdida de operaciones o datos. Se especifican políticas de recuperación ante desastres, que marcan las pautas a seguir para que la organización restaure sus operaciones e información y vuelva a funcionar con total normalidad. La continuidad del negocio es el plan de emergencia al que recurre el negocio para trabajar mientras que ciertos recursos no están operativos.
- **Educación del usuario final:** Las personas son causantes de grandes agujeros de seguridad en una red, comprometiendo así gran cantidad de datos sensibles, tanto personales como laborales (si se encuentran en el entorno de trabajo). Es imprescindible educar a las personas en buenas prácticas en lo que a seguridad cibernética se refiere, ya que pueden introducir accidentalmente un virus en un sistema seguro. Algunas de las prácticas más comunes son eliminar archivos adjuntos de correos sospechosos, o no introducir unidades USB no identificadas.

Dentro de todo el mundo de la ciberseguridad, este Trabajo pone el foco en la seguridad en redes. Dentro de una red, la seguridad debe estar garantizada desde el núcleo hasta el borde del perímetro de la red. Esto implica el uso y aplicación de una política de seguridad IT y de la implementación de hardware y software adecuado para proteger toda la infraestructura de red y todo el tráfico de ataques externos. Los principales aspectos que debe cumplir el modelo de defensa de la red son [12]:

- **Confidencialidad:** Es necesario asegurar y mantener la privacidad de los datos que atraviesan la red.
- **Integridad:** Se debe asegurar que los datos se mantienen consistentes, verificados, precisos y confiables. En otras palabras, éstos deben ser protegidos de modificaciones no autorizadas o su destrucción.
- **Disponibilidad:** Hay que asegurar que los datos son accesibles en el momento en que se necesiten. Esto implica estar preparados para posibles contratiempos como cortes de luz, fallos de hardware o cualquier otro tipo de eventos que provoquen la indisponibilidad de los datos.

Estos requisitos son aplicables tanto a redes inalámbricas como a redes cableadas. No obstante, a pesar de la comodidad de instalación de una red inalámbrica y de la ocasión de brindar movilidad a los dispositivos que se conecten a ella, una red cableada, a priori, proporcionará más seguridad debido al hecho de que para conectarse a ella, es necesario hacerlo a través de un cable físico (aun así, una buena configuración de la red Wi-Fi asegura una protección a la altura de una red cableada). La señal de una red 802.11 llega a los alrededores del entorno donde está instalada y puede ser más fácilmente atacada. Para redes de trabajo, es siempre más recomendable confiar en las redes cableadas en detrimento de las redes Wi-Fi; aunque, si se usa una red inalámbrica, hay que asegurarse de que ésta está bien configurada, confiando en protocolos de autenticación como WPA2-Enterprise con autenticación 802.1X [13].

1.1.1 Arquitectura básica del sistema de seguridad de una red corporativa

Los esquemas de seguridad de redes dependen del propósito final de éstas. No requiere la misma inversión en seguridad una red doméstica que una red corporativa de una empresa. En el primer caso, no es imprescindible invertir dinero en hardware especializado en seguridad, y es el router frontera (el que separa la red privada doméstica de Internet) el encargado de correr un software cortafuegos y de hacer NAT [14].

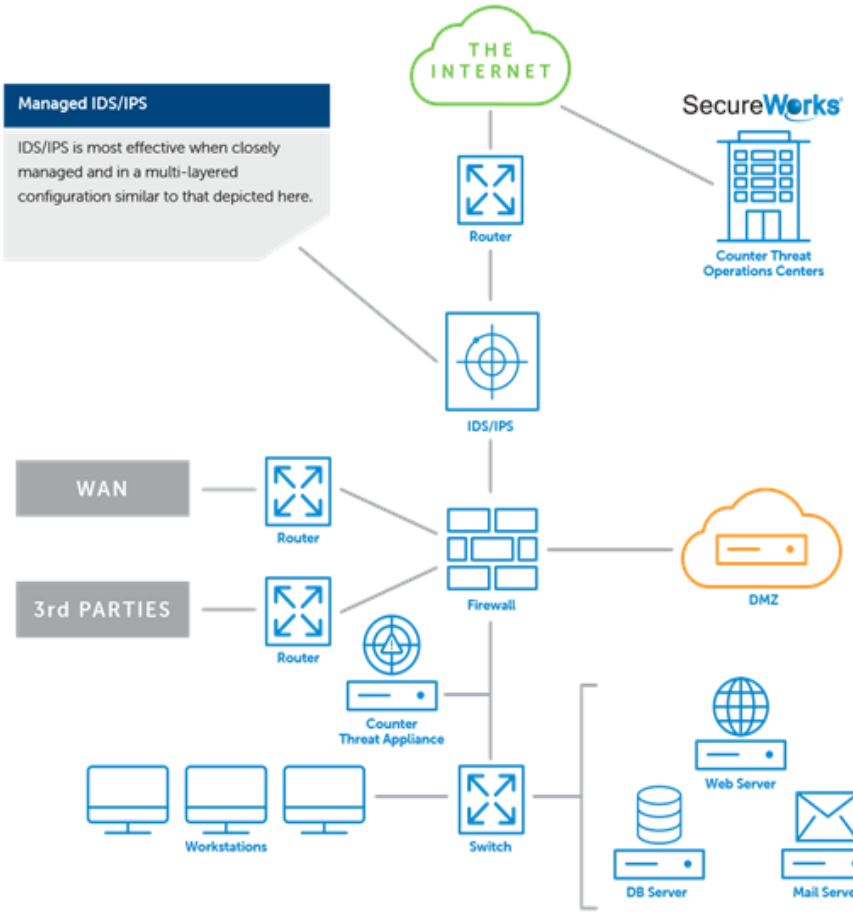


Figura 1. Ejemplo de una red segura que utiliza IDS/IPS, una red DMZ y un firewall. Fuente: SecureWorks

Para el caso de una red empresarial sí se requiere de una infraestructura más específica y compleja (véase un ejemplo de infraestructura en la Figura 1), dado que estas redes son mucho más grandes y manejan datos sensibles. Esta razón hace que los firewalls software ejecutados en los routers no proporcionen el rendimiento necesario en redes de mayor envergadura. Es aquí donde aparecen los dispositivos independientes o *standalone*. Un firewall *standalone* es un hardware exclusivamente dedicado a filtrar y rechazar el tráfico no autorizado en la red a la que protege, sin estar integrado en otro dispositivo. Estos equipos suelen venir equipados con numerosos puertos para permitir su conexión a distintos sistemas, y proporciona las capacidades demandadas por empresas con una infraestructura de red considerable.

Además de los cortafuegos, aparecen otros dos elementos relevantes en materia de seguridad: la red DMZ y los sistemas de detección de intrusos (IDS) o sistemas de prevención de intrusos (IPS). Una

zona desmilitarizada o red DMZ es una red aislada que se encuentra dentro de la red propia de la organización, y en ella se encuentran todos los recursos de la empresa que deben ser accesibles desde Internet, como el servidor web o el correo [15]. La particularidad de esta red reside en las conexiones que permite: se puede acceder a la red DMZ desde Internet o desde la propia red local, pero las conexiones desde la DMZ a la red local están bloqueadas. Esto es así porque los servidores que pueden ser accedidos desde Internet son más vulnerables a recibir ataques y, gracias a esta medida, si un hacker ataca un servidor de la DMZ tiene mucho más complicado acceder a la red local de la empresa.

La otra línea de defensa ante comportamientos maliciosos son los sistemas de prevención de intrusos (IPS) o los sistemas de detección de intrusos (IDS), siendo estos últimos objeto de estudio del presente Trabajo. Ambos actúan de forma similar en cuanto a detección de ataques; no obstante, el comportamiento de ambos posteriormente al ataque es distinto, como se desarrollará pormenorizadamente al final de este capítulo.

1.2 Clasificación de ataques a una red

Antes de proceder al lanzamiento de un ataque, un hacker intenta recopilar información acerca de las vulnerabilidades del sistema objetivo para ver cuál es la mejor forma de comprometer el sistema. El atacante escanea la red para encontrar agujeros en el sistema y posteriormente ejecutar código malicioso. Este código puede usarse para tumbar directamente la red o comprometer ciertos hosts pertenecientes a dicha red [16]. Existe un abanico bastante amplio de métodos para poner en entredicho la seguridad de una red, ya sea cableada o inalámbrica. Se destacan a continuación los ataques más comunes y relevantes a una red, categorizados en ataques basados en web o basados en el sistema [17]. De forma resumida, los ataques existentes en cada clase son:

- Ataques basados en la web: inyección SQL, *man in the middle*, ataques de diccionario y fuerza bruta, *cross-site scripting*, denegación de servicio (DoS), *phishing*.
- Ataques basados en el sistema: virus, gusanos, troyanos, *backdoor*, *exploits*, *bots*.

1.2.1 Ataques basados en la web

Este tipo de ataques son objetivo de sitios web o aplicaciones web. Dichas amenazas aprovechan los sitios web, sistemas de gestión del contenido web o componentes que alojan servicios web para hacerse con credenciales de usuarios, leer movimientos financieros de visitantes de las páginas o directamente infectar el sistema con cualquier tipo de malware [18]. Dentro de esta categoría se pueden considerar los siguientes ataques:

I. *Inyección SQL*

Los ataques de inyección son esencialmente vulnerabilidades que permiten a un ciberdelincuente inyectar código malicioso para iniciar un ataque y comprometer la seguridad de un sistema. Existen muchos tipos de ataques de inyección, aunque los más comunes son los ataques de inyección SQL. Cualquier página web que utilice una base de datos SQL puede verse comprometida por un ataque de este tipo. A pesar de existir desde hace mucho tiempo, no deja de utilizarse; a día de hoy incluso existe software especializado en inyección SQL automatizada, lo que hace más fácil el robo de datos a los ciberdelincuentes.

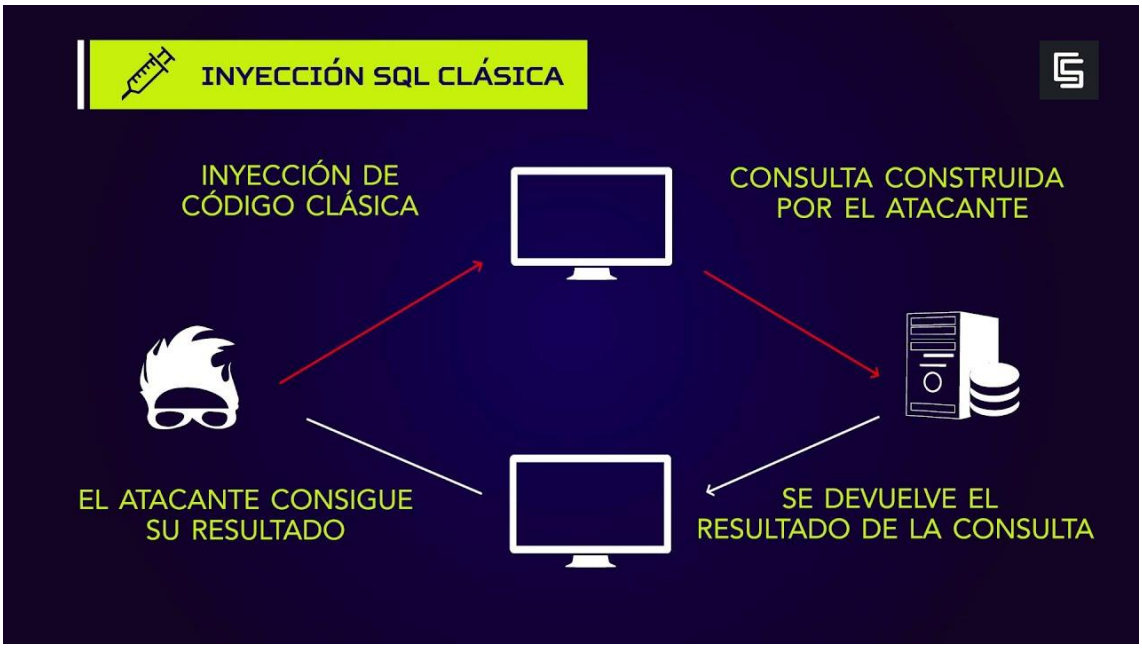


Figura 2: Esquema sencillo de realización de un ataque de inyección SQL. Fuente: Universidad Rey Juan Carlos

En un ataque de este tipo, el intruso aprovecha los cuadros de introducción de datos (nombre, contraseña, etc.) para enviar sus propias consultas SQL a la base de datos y, de esta manera, crear, modificar, leer, actualizar o borrar los datos existentes en la base de datos. Si el ataque se realiza correctamente, es imposible detectar una inyección SQL hasta el momento en el que se descubre que los datos han sido robados [19].

II. *Man in the middle*

Como bien sugiere su nombre ("hombre en el medio" si lo traducimos al español), un ataque *man in the middle* consiste en ejercer de intermediario entre dos comunicaciones y robar toda la información que se envía. En otras palabras, el ciberdelincuente "pincha" la comunicación de la víctima con un punto de acceso, por ejemplo, siendo capaz de visualizar todo el tráfico en ambas direcciones. Precisamente este último caso es uno de los modelos más habituales de este ciberataque. El hacker configura un dispositivo de forma que parezca un punto de acceso legítimo; por ejemplo, configurar un ordenador para crear una red Wi-Fi y que la víctima potencial se conecte a ella. De esta forma busca hacerse con todos los datos que el usuario envía. Esta práctica suele ser bastante frecuente en lugares públicos, donde los atacantes aprovechan las redes inalámbricas abiertas que están disponibles en estaciones de tren, centros comerciales, etc. Otro tipo de *man in the middle* se lleva a cabo en los

navegadores, donde el hacker inserta código malicioso en el sistema de la víctima y actúa como intermediario recabando todos los datos que el afectado introduce en el navegador.

Afortunadamente, se pueden tomar medidas para evitar caer en este tipo de ataques. Una de las primeras acciones es evitar las redes abiertas. En el párrafo anterior se destacaba el hecho de que estas redes son aprovechadas por los hackers. Por tanto, hay que evitar aquellas redes que estén abiertas o que tengan un cifrado débil.

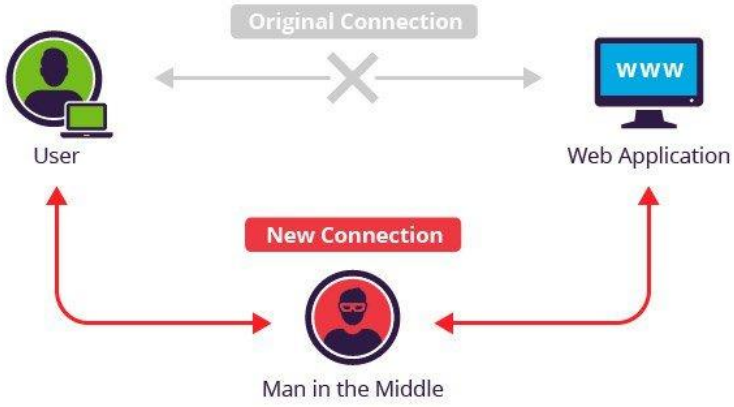


Figura 3: Esquema sencillo de un ataque Man in the Middle. Fuente: tecnonucleous.com

La navegación por páginas HTTPS (páginas web cifradas) también permite salvaguardar datos sensibles mientras se navega en redes públicas. El uso de herramientas de VPN para cifrar las conexiones y la utilización de contraseñas complejas o el procedimiento de autenticación en dos pasos también son opciones válidas para prevenir este tipo de ataque [20].

III. Ataques de diccionario y de fuerza bruta

El ataque de fuerza bruta consiste en dar con una clave probando todas las combinaciones posibles hasta encontrar aquella que provee el acceso. Esta forma de intentar dar con una contraseña puede llegar a costar mucho tiempo, incluso años, dependiendo de la complejidad y longitud de la contraseña. Existen ciberdelincuentes que tienen como objetivo un mismo sistema durante meses o años. Suele combinarse con un ataque de diccionario para agilizar la tarea de adivinar la contraseña.

A diferencia de la fuerza bruta, el ataque de diccionario utiliza una lista extensa de palabras comunes o de contraseñas vistas en sitios anteriores. El hacker confía en que el usuario use para su contraseña palabras fáciles de recordar o de uso recurrente. En la actualidad, se exigen ciertos requisitos a un usuario a la hora de crear una contraseña para dotar a ésta de la robustez necesaria para no ser presa de ataques de este tipo [21].

IV. Cross-site scripting

Esta vulnerabilidad aprovecha la confianza que el usuario tiene en un sitio web en particular, y puede ejecutarse de dos formas: de forma reflejada y de forma almacenada. En la forma reflejada, el atacante modifica valores que usa la web para pasar variables entre dos páginas. Para poder robar las cookies del usuario y, por ende, sus credenciales de acceso a la web, primero es necesario que la víctima

ejecute un determinado comando dentro de la web. Para conseguir este propósito, los cibercriminales envían correos falsos para que las víctimas entren en un enlace prácticamente idéntico al original y así puedan robar los datos.

Por otra parte, la forma almacenada consiste en insertar código HTML malicioso en los sitios que lo permitan.

Para explicar de forma más concisa el modus operandi de este ataque, se escenifica la siguiente situación:

1. Por un lado, se tiene un servidor que aloja la página web "mibanco.com". Este servidor es vulnerable a XSS.
2. Por otra parte, el ciberdelincuente consigue introducir código malicioso en la página "mibanco.com". Así, una vez que el usuario intente acceder a esta página, será redireccionado a una página exactamente igual que "mibanco.com", controlada por el atacante.
3. Finalmente, una vez que la víctima ingrese sus datos personales, no lo estará haciendo en la página original si no en la réplica. De esta forma, se ve comprometida su información personal y el hacker se hace con los datos sensibles de la víctima.

Si un sitio web es vulnerable a XSS un hacker puede ejecutar scripts en la página web, que pueden provocar desde la redirección hacia otro sitio para robar información, hasta descargar malware en el equipo de la víctima y que se ejecute en el sistema, pudiendo tomar el control del dispositivo. Si no se realizan controles en el sitio web para evitar la ejecución de comandos desde la propia página, los ciberdelincuentes aprovechan esta brecha de seguridad para actuar maliciosamente [22].

V. Denegación de servicio (DoS)

Un ataque de denegación de servicio ocurre cuando los usuarios no pueden utilizar los recursos de una red, sistemas de información u otros dispositivos, debido a la acción maliciosa de un ciberdelincuente. Los servicios tumbados pueden ser de distinta índole: sitios web, cuentas bancarias, u otros servicios. La ofensiva resulta exitosa cuando se inunda la red o el host objetivo hasta que éste no puede responder y se acaba bloqueando, no permitiendo el acceso de usuarios legítimos al servicio. Los ataques DoS pueden resultar muy costosos tanto en tiempo como en dinero a una empresa mientras que sus servicios están tumbados. Este tipo de ataques también se pueden ejecutar de forma distribuida, es decir, con un grupo de máquinas operando de forma conjunta contra un objetivo común.

Los indicios de un posible DoS se asemejan a problemas de disponibilidad o problemas técnicos. La mejor manera de detectar e identificar un ataque de este tipo es monitorizando y analizando el tráfico de la red a través de un IDS o un *firewall*, e incluso se pueden establecer reglas de tráfico que den una alarma cuando se detecte un volumen de tráfico anormal o descartar paquetes que cumplan con ciertos criterios. En el caso de estar siendo asediado por un ataque de denegación de servicio, se recomienda no perder de vista otros recursos o servicios que formen parte de la misma red. Comúnmente, los ciberdelincuentes suelen lanzar ataques de denegación de servicio y al mismo tiempo efectuar otros ataques para comprometer otros servicios de la red [23].

VI. ***Phishing***

Este ataque guarda bastante relación con el *cross-site scripting* descrito anteriormente. De hecho, el aprovechamiento de las vulnerabilidades XSS es bastante común para realizar *phishing*. Este delito se basa en engañar a la víctima para conseguir información sensible, como contraseñas, datos bancarios, etc. Existen muchas formas de realizar esta suplantación, pero la más extendida es hacerlo a través de un correo electrónico que imita la estructura de un correo legítimo enviado por cualquier organización. El contenido de este mail intenta intimidar a la víctima, exigiendo que vaya a un sitio web para solucionar un problema o será penalizado de algún modo. Si la víctima pincha en el enlace, será redirigido a un sitio web que es una copia del original, y si ésta se registra con su usuario y contraseña, la información de inicio de sesión pasará a estar en manos del atacante [24].

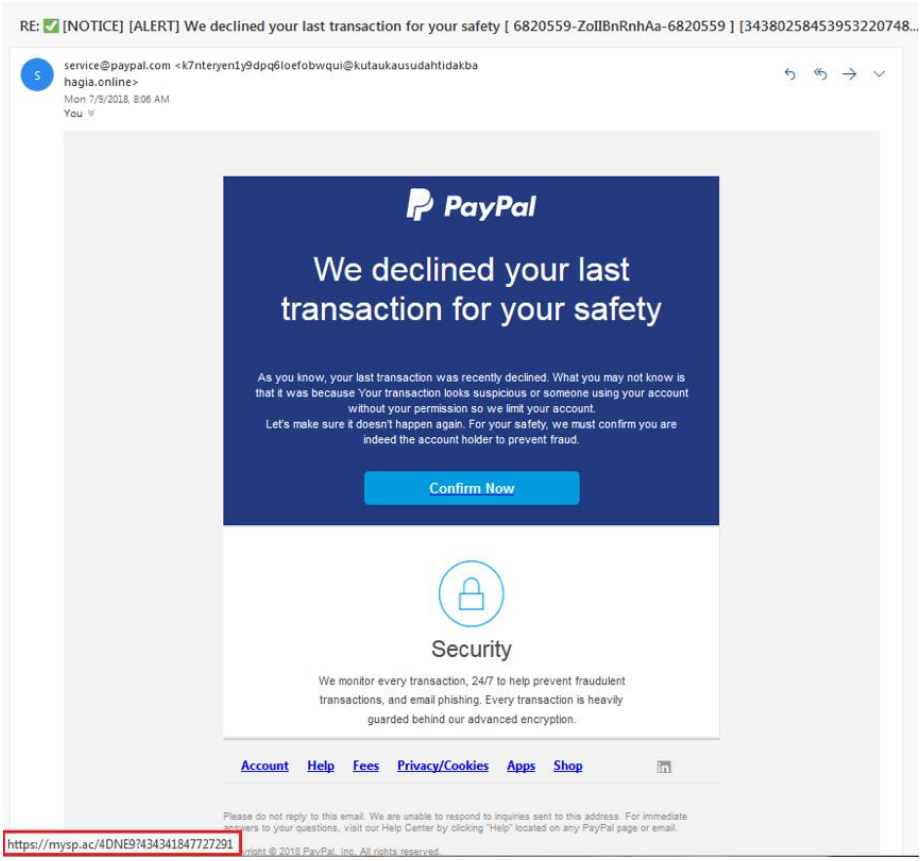


Figura 4: Ejemplo de phishing, intentando replicar un correo de PayPal. Fuente: malwarebytes.com

Este tipo de ciberataque no necesita de unos conocimientos específicos para ser llevado a cabo; de hecho, se considera la forma de ciberataque más sencilla y una de las más efectivas, debido a que depende directamente de una persona, con independencia de lo seguro que sea su equipo. Reconocer un ataque de este tipo puede no ser fácil en muchas ocasiones, pero existen señales que pueden indicar que se está ante un correo malicioso:

- En el correo electrónico se indica que se ha ganado un premio, o similar.
- Se insta en el correo a que se acometa una acción y, en el caso de que no se haga, se penalizará al usuario de alguna forma.
- El correo contiene archivos adjuntos extraños. Dichos archivos pueden contener malware.

- El enlace adjunto en el correo electrónico contiene fallos ortográficos, o resulta extraño. Antes de hacer clic en el enlace, se recomienda pasar el cursor por encima del enlace para ver la URL real en lugar de hacer clic directamente en el enlace incorporado.

1.2.2 Ataques basados en el sistema

Estos ataques están destinados a comprometer un equipo en concreto o una red de equipos. Gran parte de los ataques de este apartado se podrían considerar como subtipos de virus informáticos. Sin embargo, existen ciertos matices entre ellos, y algunos son tan habituales que resulta adecuado dedicar individualmente un apartado para cada uno y mencionarlos con algo más de profundidad. Dentro de esta tipología se encuadran los siguientes ataques:

I. Virus

Los virus informáticos son programas maliciosos cuyo objetivo es "infectar" los archivos de un sistema con la intención de dañarlos o modificarlos. Normalmente, un virus entra en un equipo a través de un archivo ejecutable (que es el que contiene el código malicioso) y, una vez ejecutado, el virus infecta el equipo de la víctima.

Los daños efectuados al software son variados y con distinto fin: el atacante puede solamente realizar una broma sin afectar ningún archivo del sistema, o puede borrar o dañar archivos críticos para el funcionamiento de éste. Los antivirus y los cortafuegos son esenciales para prevenir nuestros equipos ante una posible infección de algún tipo de virus [25].

II. Gusanos

A diferencia de un virus normal, un gusano informático hace réplicas de sí mismo y las guarda en diferentes ubicaciones de un equipo. La meta de este tipo de malware es inundar los ordenadores y las redes informáticas hasta su bloqueo total. A diferencia de los virus, los gusanos no infectan archivos e intentan propagarse hasta afectar al mayor número de equipos posibles. Después de replicarse en el ordenador afectado, distribuye posteriormente las copias a través de correos electrónicos o programas de descarga *peer to peer* [26], a diferencia de los virus; éstos últimos no se propagan automáticamente, sino que es necesario que el archivo infectado viaje hacia otra víctima potencial.

III. Troyanos

Un troyano o caballo de Troya es un malware que toma su nombre de la famosa historia del caballo de Troya mencionada en la Odisea de Homero. Toma la apariencia de un software totalmente legítimo, pero que esconde código malicioso capaz de dañar, robar, o infligir cualquier tipo de acción perniciosa en los datos de un equipo. Este tipo de ataque busca engañar a la víctima para que ejecute el archivo donde se encuentra el código malicioso, y una vez ejecutado, el troyano realiza la función para la que fue diseñado: desde molestar al usuario abriendo ventanas hasta robar datos o dañar al host. También es sabido que los troyanos abren puertas traseras (*backdoor*) para permitir el acceso a otros usuarios al sistema atacado.

La disparidad existente entre los troyanos y los otros dos tipos de ataques vistos anteriormente es que los primeros no se reproducen infectando otros archivos ni se autorreplican. Es indispensable la colaboración del usuario para que el ataque tenga éxito [27].

IV. Backdoor

La puerta trasera es una forma secundaria de acceder a un sistema, evitando los mecanismos de autenticación normales. Algunos backdoors son colocados expresamente por los programadores de un software, mientras que otros son generados por malware como virus o troyanos. Estas puertas traseras son utilizadas por los atacantes para entrar de forma más rápida a un sistema después de haber sido comprometido [27].

V. Exploits

Un exploit es el agujero de seguridad existente en un sistema que podría ser aprovechado por un hacker para su beneficio propio. Por esta razón no se considera expresamente como código malicioso. Dicho con otras palabras, es la llave para que un ciberdelincuente acceda a un sistema, y posteriormente ejecute acciones maliciosas como la infección de archivos o robo de datos sensibles.

Existen dos tipos de exploits: los conocidos o los *zero-day*. Los conocidos son aquellos de los que se tiene conocimiento y se pueden tomar medidas para evitar comprometer la seguridad de un sistema. Por otro lado, los *zero-day* (o los exploits desconocidos) se caracterizan por aprovechar vulnerabilidades de las que todavía no se tienen constancia. Cuando se usan no suelen existir medidas que bloqueen el ataque y eso los hace indetectables. Para evitar este tipo de ataques, se recomienda tener una política de actualizaciones eficaz [28].

VI. Bots

Los bots no tienen por qué ser usados con fines maliciosos. Con un buen propósito, un bot se usa para automatizar tareas y proporcionar información, en lugar de que una persona sea la encargada de realizarlas. En contraposición, un bot usado por un ciberatacante está diseñado para infectar un equipo y conectarlo a un servidor central que es el encargado de comandar toda la red de dispositivos infectados con el mismo bot (botnet). Con una red de bots, se pueden lanzar ataques de denegación de servicio contra un objetivo.

Los bots poseen todas las ventajas de los gusanos, pero suelen ser mucho más versátiles a la hora de infectar equipos, y suelen modificarse rápidamente una vez se hace pública una vulnerabilidad en un sistema. Aprovechan las puertas traseras abiertas por gusanos y virus, infectando las redes de una forma silenciosa [27].

1.2.3 Otra clasificación de ataques

La clasificación que se incluye a continuación no suele ser tan común como la anteriormente planteada. Sin embargo, es la existente en la base de datos UNSW-NB15, que es la que se usará posteriormente para testar los algoritmos desarrollados; de ahí la necesidad de presentar esta otra clasificación. UNSW-NB15 contiene 9 agrupaciones de ataques distintas, algunas ya descritas en los apartados anteriores (*worms*, *exploits*, *backdoor*, DoS). El resto de clases son las que siguen [29]:

- **Shellcode:** Ataque en el que el hacker ejecuta una porción de código en un shell para controlar la máquina donde este se está ejecutando.
- **Fuzzers:** Ataque consistente en inundar a un programa o sistema de datos inesperados, inválidos o aleatorios con el fin de descubrir fallas de seguridad en el sistema.
- **Analysis:** Esta tipología recoge una gran variedad de intrusiones que penetran en el sistema a través de aplicaciones web aprovechando puertos abiertos, correos electrónicos y scripts web. Las inyecciones SQL y el Cross-Site Scripting descritos en el Apartado 1.2.1 entrarían dentro de esta categoría.
- **Generic:** Este ataque se centra en la criptografía, y en esta familia de ataques se podrían incluir los ataques de fuerza bruta, por ejemplo. Un ataque genérico es aquel que se puede ejecutar sin tener en cuenta la configuración del cifrado.
- **Ataque de reconocimiento (reconnaissance attack):** Se basa en reunir información sobre una red para localizar sus posibles brechas de seguridad y lanzar un ataque por ellas.

1.3 Sistemas de detección de intrusos (IDS) y sistemas de prevención de intrusos (IPS)

Para ser capaces de detectar los ataques enumerados en el punto anterior, se hace necesario conocer sus características y el comportamiento de éstos en la red. Para realizar esta tarea, el administrador de la red precisa de un sistema de monitoreo para ser capaz de visualizar las diferencias existentes entre el tráfico malicioso y el normal. Es aquí donde entran en escena los sistemas de detección de intrusos y los sistemas de prevención de intrusos. Ambos sistemas de seguridad se encargan de vigilar el tráfico examinando la red y los puertos, y reconociendo patrones sospechosos. Un ataque puede ser detectado de diversas formas; desde un aumento inesperado del volumen de tráfico en la red o el volumen de tráfico en función del encabezado de los paquetes. Aun así, para detectarlos hace falta procesar grandes flujos de datos casi en tiempo real [16]. Precisamente por el manejo y análisis de grandes cantidades de datos los algoritmos de *machine learning* se han colocado como pieza fundamental de este tipo de sistemas, como se verá en capítulos posteriores.

Diseñar un mecanismo de seguridad en tiempo real que sea capaz de identificar cualquier tipo de ataque es un reto enorme. Prácticamente todos los métodos de detección de ataques actuales necesitan de información previa para poder usarla durante la detección de los ciberataques. A la hora de diseñar un sistema de detección de intrusos o un sistema de prevención de intrusos, se debe tener precaución al realizar cualquier suposición o al elegir las características adecuadas del tráfico que son más representativas para detectar los ataques. Seguidamente, se tratan en mayor profundidad las diferencias existentes entre ambos mecanismos de seguridad, y se exponen las distintas formas de clasificación de los IDS y los IPS.

1.3.1 Sistemas de detección de intrusos (IDS)

Los IDS contienen una base de datos actualizada constantemente con muchos datos recogidos de ataques para poder identificarlos. Esta solución se basa en monitorizar el tráfico entrante en la red a través de un escaneo profundo de puertos y un análisis de red, y a su vez va comparando dicha información con la contenida en la base de datos de ataques. Ante cualquier movimiento sospechoso, este sistema de detección emite una alerta a los administradores del sistema, responsables de tomar las medidas pertinentes [30]. Teóricamente, se presentan diversas formas de clasificar los IDS atendiendo a diversos criterios:

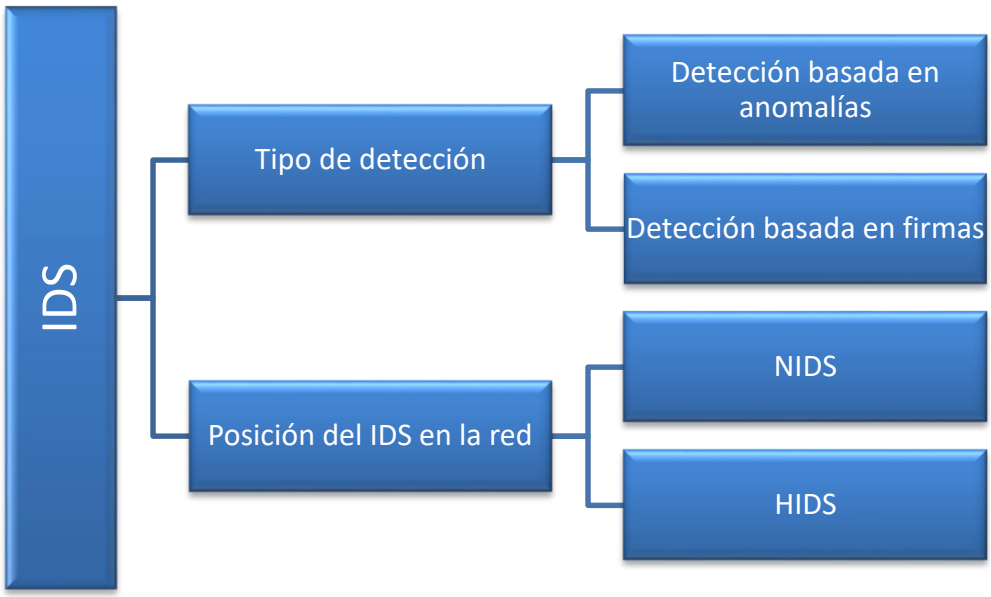


Figura 5: Clasificación de los IDS atendiendo a distintas categorías.

1.3.1.1 Clasificación según el tipo de detección [31]

- *Detección basada en anomalías:* Estos IDS funcionan comparando el tráfico registrado con un patrón de datos etiquetados como "tráfico normal". Si se identifica una desviación significativa entre el modelo de comportamiento normal y el tráfico analizado, se marca el tráfico como malicioso. Este tipo de sistema es capaz de detectar ataques que resultan desconocidos, pero en contraposición produce falsas alarmas.

- *Detección basada en firmas:* Funcionan de forma parecida a un antivirus. Emplean una firma (patrón que corresponde a una amenaza ya conocida y que está registrada en una base de datos de ataques) de ataques ya registrados, y si el tráfico registrado coincide con una de las entradas de la base, se emite una alerta. Un ejemplo de IDS de este tipo es Snort, un software de código abierto que monitoriza la red comparando cada paquete que observa con un conjunto de reglas. Este tipo de IDS no puede detectar ataques que no se encuentren registrados en la base de datos, por lo que no suelen dar falsos positivos.

1.3.1.2 Clasificación según la posición del IDS en la red [31]

- *Host-based Intrusion Detection System (HIDS):* Monitoriza el tráfico entrante en una sola máquina, y recoge datos de ella como logs del sistema y uso de los recursos.

- *Network-based Intrusion Detection System (NIDS):* Monitoriza una red completa y analiza el tráfico que fluye por ella. A diferencia de los HIDS, el despliegue de un nuevo dispositivo en la red no requiere de cambios en el sistema de detección de intrusos. Es mucho más eficiente actualizar un NIDS que tener que actualizar en cada uno de los equipos el HIDS. Dentro de los NIDS, hay dos formas de analizar los datos recabados en la red: basado en paquetes y basados en flujo. Los NIDS basados en paquetes tienen que analizar todo el contenido de la carga del paquete, incluidas las cabeceras, mientras que en los basados en flujo se analizan agregados de tráfico, por lo que la cantidad de datos a analizar es más reducida. Este último tipo proporciona información y patrones sobre el tráfico, mientras que los NIDS basados en paquetes facilitan información útil para detectar ataques.

1.3.2 Sistemas de prevención de intrusos (IPS)

Los IPS nacieron como una extensión de los IDS, y, a diferencia de estos últimos, están diseñados para actuar de manera proactiva: se encargan de detectar la actividad maliciosa y de detenerla. El IPS monitoriza de manera continua el tráfico en una red, protegiendo a todos los dispositivos de ésta de ataques. Es capaz de tomar decisiones sobre el control de acceso basadas en el tráfico monitorizado; según la forma de detección, un IPS puede detectar comportamientos maliciosos comparando los flujos monitorizados con una base de datos (como un antivirus) o hacerlo basándose en anomalías, tomando como referencia un comportamiento normal de tráfico [30]. Se pueden distinguir los siguientes sistemas de prevención de intrusos [32]:

I. *Basados en red LAN (NIPS):* Monitorizan la red cableada buscando tráfico sospechoso analizando la actividad por protocolos de comunicación LAN.
II. *Basados en redes inalámbricas (WIPS):* Monitorizan la red inalámbrica buscando tráfico sospechoso analizando la actividad por protocolos de comunicación inalámbrica.
III. *Análisis de comportamiento de red (NBA):* Examinan el tráfico de red para identificar amenazas que generan tráfico inusual.
IV. *Basados en host (HIPS):* Monitorizan un único host en busca de actividad sospechosa.

Después de presentar de forma concreta qué es la ciberseguridad y los sistemas de detección de intrusos, y hacer un repaso de los ataques más frecuentes a una red de telecomunicaciones, en el siguiente capítulo se expondrá la relación existente entre los IDS y la inteligencia artificial, presentando los distintos tipos de clasificadores que podrían constituir un sistema de detección de intrusos y los pasos a seguir para el diseño de un modelo adecuado a este tipo de sistemas.

2 INTELIGENCIA ARTIFICIAL: UN NUEVO IMPULSO A LA CIBERSEGURIDAD

Desde las recomendaciones de películas que nos proporciona Netflix hasta avances en campos tan importantes como la medicina, entre otros muchos, son consecuencia directa de la inteligencia artificial. Justamente, uno de los muchísimos sectores que se ha visto beneficiado por este crecimiento ha sido la ciberseguridad. La IA, a diferencia de las técnicas tradicionales, puede analizar el comportamiento normal de los usuarios y del tráfico y reconocer de esta manera comportamientos anómalos que podrían ser indicios de un ataque, evitando estas ofensivas incluso antes de que sean reportadas y solucionadas oficialmente [33].

Concretamente, este capítulo versa sobre uno de los subconjuntos más relevantes de la inteligencia artificial: el aprendizaje máquina o *machine learning*. Se exponen en un primer apartado las diferencias entre los conceptos de inteligencia artificial, aprendizaje máquina y aprendizaje profundo. Posteriormente, se enumeran los tipos de métodos de aprendizaje máquina más comunes para tareas de clasificación (en este caso, clasificación de ciberataques), y se presenta el procedimiento para diseñar un algoritmo de este tipo al completo. Para finalizar, se recogen en un último apartado las métricas más comunes para evaluar el rendimiento del método a la hora de clasificar.

2.1 Inteligencia artificial, machine learning y deep learning

Según Pariwat Ongsulee, se podría definir brevemente la inteligencia artificial como "la inteligencia mostrada por las máquinas" [34]. Ampliando un poco más esta definición, el término "inteligencia artificial" se aplica cuando una máquina replica comportamientos cognitivos propios de un ser humano, tales como resolver problemas o aprender.

A pesar de estar presente en la industria desde la década de 1980 aproximadamente, el paso adelante de la IA no llegó hasta la década de los 90, con la introducción de los métodos probabilísticos y bayesianos en el aprendizaje automático. Los avances en los algoritmos y el incremento masivo de datos digitales sobre todo en esta última década y media han terminado de confirmar que la IA resulta especialmente útil para manejar y generar conocimiento de cualquier fuente de datos digital [35]. Las materias de investigación de la IA incluyen el procesado del lenguaje humano, la habilidad de manipular objetos, el razonamiento, aprendizaje, conocimiento y posterior planificación en base a lo aprendido, entre otros, y, a largo plazo, hacer que las máquinas adquieran un grado completo de inteligencia (adquieran consciencia), lo que se conoce como IA fuerte. En la actualidad, las máquinas pueden reproducir comportamientos humanos, pero sin consciencia (IA débil) [34].

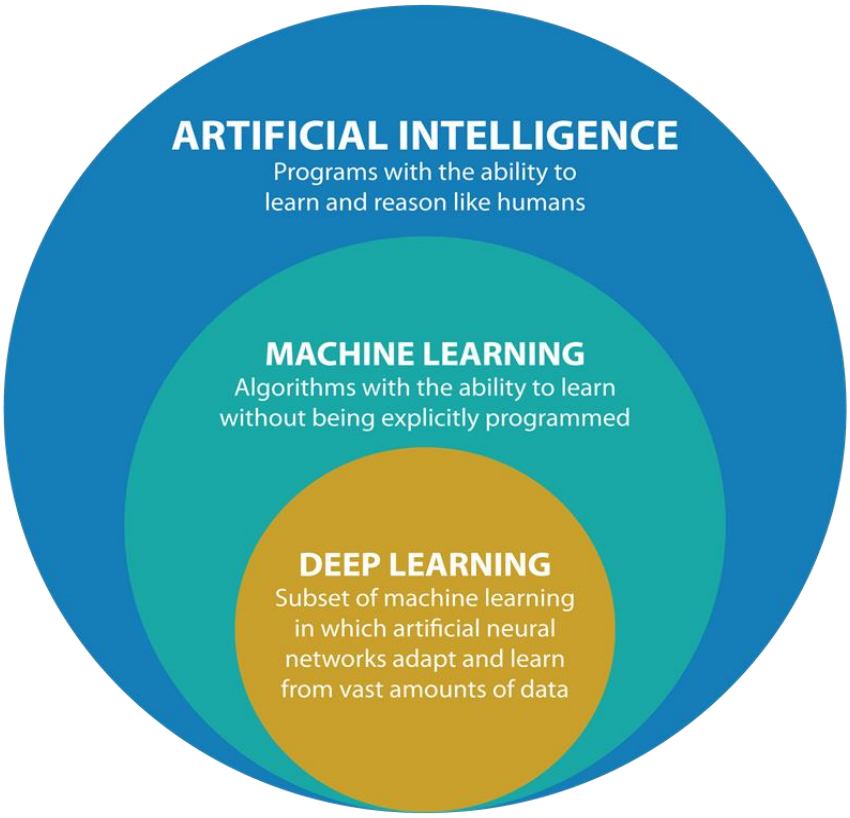


Figura 6: Organización jerárquica de la inteligencia artificial, el aprendizaje máquina y el aprendizaje profundo. Fuente: medium.com

Uno de los subcampos de la inteligencia artificial es el aprendizaje máquina o *machine learning*, que surgió a raíz del estudio del reconocimiento de patrones y la teoría de aprendizaje computacional en inteligencia artificial. El ML tuvo su nacimiento como tal en la década de los 90, aprovechando todos los avances conseguidos en IA, y ha acabado por convertirse en el subcampo más popular y exitoso de la IA [36]. El aprendizaje máquina se ocupa del desarrollo de algoritmos que pueden aprender de y realizar predicciones en base a ingentes bases de datos, construyendo un modelo de aprendizaje gracias a estos últimos. En relación a esta última idea, un término que aparece íntimamente relacionado con el ML es la minería de datos, o *data mining*, que consiste precisamente en aplicar algoritmos de aprendizaje a grandes bases de datos con el objetivo de buscar tendencias o patrones en todo el conjunto [37]. Una definición alternativa propuesta por Ethem Alpaydin en [37] afirma que "el *machine learning* consiste en programar ordenadores para optimizar un criterio de funcionamiento usando datos o experiencias pasadas". Se tiene un modelo definido por parámetros, y la acción de aprender consiste en optimizar dichos parámetros usando los datos o experiencias anteriores.

El ML se apoya en gran parte en la estadística para construir sus modelos matemáticos, y en la informática. Justamente, este último campo de conocimiento juega un papel doble en el ML: primeramente, se necesitan algoritmos eficientes para solucionar el problema de optimización de parámetros, así como para almacenar y procesar la enorme cantidad de datos que se acostumbran a manejar en problemas de aprendizaje. Segundo, una vez que el algoritmo ha aprendido, la inferencia de

nuevos datos debe ser eficiente también. Dependiendo de la finalidad del algoritmo, puede ser que la eficiencia (tiempo computacional y espacio de memoria empleado) sea tan crucial como la precisión del algoritmo a la hora de inferir nuevos datos. Los modelos de aprendizaje máquina ayudan a encontrar soluciones a problemas de diversa índole: desde reconocimiento de voz o robótica hasta el análisis de grandes cantidades de datos en campos como la astronomía o la biología.

Dentro del machine learning encontramos el aprendizaje profundo o *deep learning*, un nuevo paradigma de aprendizaje a partir de datos basado en el estudio de capas sucesivas de representación de los datos cada vez más significativas. El término "profundo" no indica que el algoritmo realice una comprensión más profunda de la información que maneja, sino que representa esta idea de capas consecutivas de representaciones. La cantidad de estas capas que tiene que manejar el modelo se denomina profundidad del modelo. La profundidad de un modelo puede oscilar desde unas pocas capas hasta decenas o cientos de ellas (como por ejemplo la red GoogLeNet con 22 capas)[1] y todas ellas aprenden de forma autónoma al manejar los datos de entrenamiento. Esta estructura jerárquica de capas consecutivas y apiladas unas encima de otras se conoce en DL como red neuronal, como se verá con mayor detalle en capítulos posteriores de este trabajo [36].

Aunque las primeras ideas sobre aprendizaje profundo aparecieron en torno a 1980, no ha sido hasta hace pocos años cuando verdaderamente se ha explotado el potencial de este campo de conocimiento, principalmente por dos motivos [38]:

- El *deep learning* requiere grandes cantidades de datos etiquetados. Por ejemplo, para el entrenamiento de un vehículo autónomo se necesitan millones de imágenes y miles de horas de vídeo.

- El *deep learning* requiere una enorme potencia computacional. Este problema se ha solucionado con la llegada de las GPU que permiten realizar una ingente cantidad de operaciones matemáticas en paralelo, lo cual resulta eficiente para el aprendizaje profundo. En combinación con clusters o con el cálculo en la nube, esto permite a los equipos de desarrollo reducir el tiempo necesario para el entrenamiento de una red de *deep learning* de forma drástica.

Las aplicaciones de este campo se utilizan en sectores tan dispares como la conducción autónoma o la investigación médica. En el primer caso, el *deep learning* sirve para detectar automáticamente elementos como señales de tráfico, semáforos o peatones, ayudando así a la disminución de accidentes de tráfico. En lo referente a los avances médicos, se experimenta con técnicas de DL para realizar diagnósticos eficaces y tempranos de enfermedades potencialmente mortales, lo cual podría ayudar a mejorar la atención médica en zonas donde los facultativos escasean [39].

2.2 Clasificaciones de algoritmos de ML

La cantidad de algoritmos existentes de ML y el propósito final para el que se usan ha obligado a los investigadores a categorizarlos de distintas maneras. Los criterios de división que se presentan a continuación no son mutuamente excluyentes, y, dependiendo del problema a solventar, dichas pautas se podrían combinar de distintas formas. Los tres tipos de categorización de los modelos de aprendizaje máquina y sus subclases se describen en los posteriores apartados [40].

[1] *GoogLeNet es una red neuronal convolucional desarrollada por Google, en colaboración con otras instituciones.* https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/43022.pdf

2.2.1 Aprendizaje supervisado, no supervisado, semisupervisado y por refuerzo

Los métodos de ML se pueden clasificar de acuerdo a la cantidad de supervisión humana que reciben durante la fase de entrenamiento. Existen 4 amplias clases:

I. Aprendizaje supervisado

En este tipo de aprendizaje, los datos que se pasan al método para entrenarlo contienen las soluciones deseadas, llamadas etiquetas. Dos tareas típicas del aprendizaje supervisado son la clasificación (se entrena al modelo con las muestras etiquetadas y cuando recibe una muestra desconocida, es capaz de encasillarla en la categoría correcta según lo aprendido previamente) y la regresión (se predice un determinado valor (normalmente numérico) dado un set de características conocido como predictores).

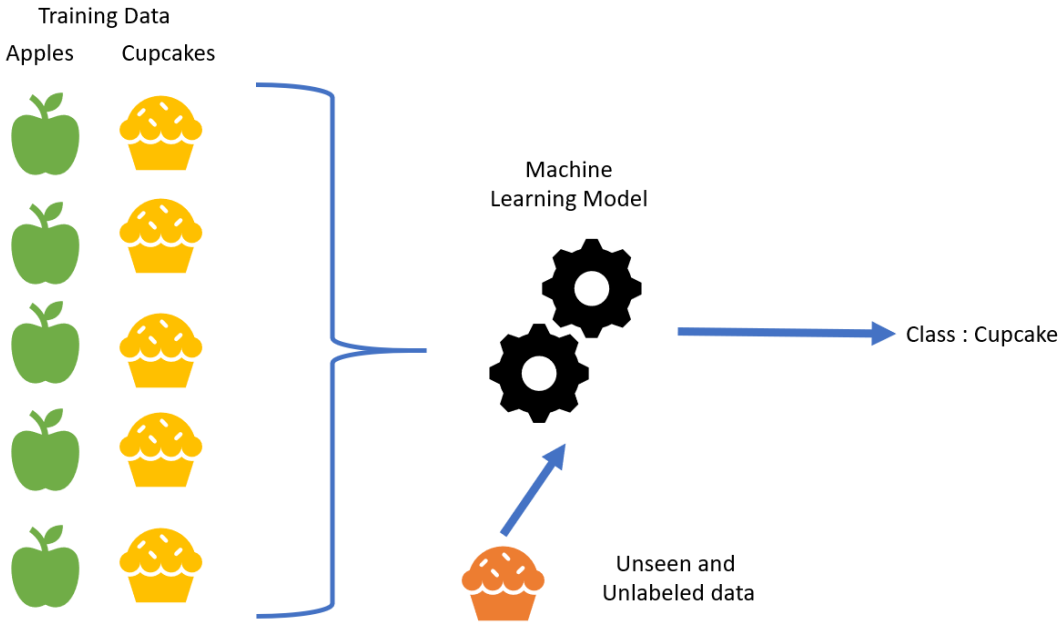


Figura 7: Esquema básico que ejemplifica la tarea de clasificación de un algoritmo de ML. Fuente: medium.com

Algunos de los algoritmos más importantes de aprendizaje supervisado para tareas de regresión son los regresores polinómicos ajustados por mínimos cuadrados (como los polinomios de Volterra), estimadores de mínimo error cuadrático medio ó procesos gaussianos para regresión (CGPR). Para tareas de clasificación se usan el "k-nearest neighbors" (KNN), las máquinas de soporte vectorial (SVM), los árboles de decisión y los bosques aleatorios, entre otros Teniendo en cuenta que la problemática de un sistema de detección de intrusos consiste en determinar si el tráfico monitorizado es indicio de un ciberataque o no, esto es, un problema de clasificación, se profundizará teóricamente en estos últimos algoritmos en puntos posteriores.

II. Aprendizaje no supervisado

En el aprendizaje no supervisado, los datos que se proporcionan al método no están etiquetados, y el sistema aprende "a ciegas". Entre las labores típicas del aprendizaje no supervisado se cuentan [40]:

- ***Clustering:*** consiste en realizar grupos de instancias con características similares. Entre los métodos utilizados en este apartado se sitúan el k-Means y el EM ("Expectation Maximization").

- ***Visualización de los datos:*** permite representar cómo se distribuye la información y así poder detectar posibles patrones en ella. Aquí se recogen algoritmos como el análisis de componentes principales (PCA) y el t-SNE ("t-distributed Stochastic Neighbor Embedding").

- ***Reducción de dimensionalidad:*** su objetivo es simplificar los datos perdiendo la mínima cantidad de información posible. Los métodos de visualización son también aplicables a esta clase.

- ***Aprendizaje de reglas de asociación:*** se manejan grandes cantidades de datos y se intentan establecer relaciones entre ellos. El método A priori y el Eclat son dos de los más destacados.

III. Aprendizaje semisupervisado

Si se pasan al método una gran parte de datos no etiquetados y una pequeña porción de datos etiquetados estamos hablando de aprendizaje semisupervisado. La aplicación Google Photos funciona de esta forma cuando se suben fotos a su servicio. Previamente, el modelo ha sido entrenado para detectar caras en imágenes. Una vez que un usuario sube fotos a la aplicación, el algoritmo detecta automáticamente la cara de una persona A en algunas fotos, la cara de una persona B en otras, y así sucesivamente. Esta es la parte no supervisada del algoritmo (clustering). Posteriormente, pregunta quién es esa persona para etiquetarla convenientemente en todas las fotos en las que aparezca.

La mayoría de métodos encasillados en esta categoría son una mezcla de algoritmos de aprendizaje supervisado y no supervisado, como las redes de creencia profunda ("deep belief networks") [40].

IV. Aprendizaje por refuerzo

El aprendizaje por refuerzo se desmarca de la metodología de las anteriores categorías, aplicando las bases de la psicología conductista al método de aprendizaje de una máquina. Uno de los conceptos sobre los que gira esta teoría es el de condicionamiento operante, esto es, un proceso de aprendizaje por el cual una acción desemboca en algo deseable (haciendo más probable que el individuo repita dicha acción) o algo no deseable (evitando que se vuelva a ejecutar). Así pues, para conseguir estas recompensas, el ser humano recurre a técnicas de prueba y error para calcular nuevas estrategias de acción.

Este mismo procedimiento se aplica entonces a los algoritmos de aprendizaje máquina. Aprovechando las potentes capacidades de cálculo de las máquinas y que, a diferencia de un ser humano, éstas no se cansan, se pueden conseguir estrategias prácticamente perfectas [41].

2.2.2 Aprendizaje online y aprendizaje por lotes

Otro criterio de clasificación de los métodos es si éstos son capaces de aprender incrementalmente a partir de un flujo continuo de datos o no. Dentro de esta nueva tipología podemos discernir entre aprendizaje online o aprendizaje por lotes [40].

I. Aprendizaje por lotes

Esta metodología no permite el aprendizaje incremental, sino que éste debe de hacerse con los datos disponibles. Normalmente esto consume mucho tiempo y recursos, por lo que suele hacerse offline. Primeramente, el algoritmo es entrenado y después es lanzado a producción sin que aprenda nada más; el método aplica solamente lo que ha aprendido. Si se necesita incorporar nuevo conocimiento al método, debe de volver a entrenarse con todas las muestras otra vez (incluidas las nuevas), y reemplazar el modelo antiguo por el nuevo.

Afortunadamente, en la actualidad se puede automatizar este proceso de forma bastante fácil. Sin embargo, se tarda un tiempo sustancial en entrenar el nuevo modelo con toda la cantidad de datos existente. Si se necesita un modelo que se adapte rápidamente a los cambios de las muestras, sería más adecuado adoptar un enfoque de aprendizaje online.

II. Aprendizaje online

En este tipo de aprendizaje, el método se entrena recibiendo continuamente datos, bien individualmente o agrupados en lotes. Cada paso de aprendizaje es rápido y barato, por lo que el sistema puede aprender sobre la marcha. Es bastante oportuno adoptar esta estrategia cuando el sistema suele recibir los datos como un flujo continuo y debe adaptarse o cambiar rápidamente, así como cuando se tienen restricciones de almacenamiento, ya que cuando el algoritmo ha aprendido lo referente a las nuevas instancias puede descartarlas.

Un parámetro importante de estos sistemas es la tasa de aprendizaje, la cual regula la rapidez con la que el sistema se adapta a los cambios que detecta en las instancias. Si este valor es muy alto, el sistema se adaptará con premura a los cambios, a costa de olvidar rápidamente los datos antiguos. Por el contrario, un valor demasiado bajo hará que el sistema aprenda de forma mucho más lenta, pero se verá menos afectado por el ruido que puedan introducir las nuevas muestras.

2.2.3 Aprendizaje basado en instancias y aprendizaje basado en el modelo

Para finalizar, otra forma de categorizar los sistemas de ML es tomando como referencia la forma de generalizar que tienen. Con generalizar se entiende el hecho de que, dados unos datos de entrenamiento, el sistema debe ser capaz de extender lo aprendido a muestras que no ha visto nunca. La meta última de un algoritmo de aprendizaje máquina es tener un buen desempeño en las nuevas muestras. Los dos enfoques que se pueden encontrar en este apartado son el aprendizaje basado en las instancias y el aprendizaje basado en el modelo [40].

I. Aprendizaje basado en instancias

En este caso, el algoritmo aprende los ejemplos y cuando llega una instancia desconocida, la identifica midiendo la similitud entre la instancia nueva y las aprendidas en la fase de entrenamiento.

II. Aprendizaje basado en el modelo

Otra forma de generalizar partiendo de las muestras de entrenamiento es construir un modelo de dichas instancias y usarlo para realizar predicciones.

2.3 Algoritmos de clasificación

Como se comentó en anteriores apartados, el problema de detección de ataques por parte de un sistema de detección de intrusos es una tarea de clasificación. Los métodos clásicos son ampliamente usados por los investigadores, ya que se han estudiado durante décadas y, por ende, están más asentados y explorados. Sin embargo, el número de estudios relativos a los sistemas de detección de intrusos y basados en DL ha aumentado rápidamente en los últimos años. También se utilizan técnicas combinadas que unen uno o varios de estos métodos. Se hace una revisión breve del funcionamiento de algunos de estos modelos a continuación [42].

2.3.1 Naïve Bayes

Se basa fundamentalmente en la probabilidad condicional y en la hipótesis de independencia de los atributos o características. Para cada instancia, el clasificador ingenuo de Bayes calcula las probabilidades condicionales para las diferentes clases. La muestra se clasifica según la siguiente fórmula [43]:

$$\hat{y} = \underset{k\in\{1\dots K\}}{\operatorname{argmax}}\, p(C_k) \prod_{i=1}^{n} p(\, x_i \mid C_k) \qquad (1)$$

donde $\hat{y}$ representa la muestra una vez clasificada, x se refiere a la observación, C_k es la clase k-ésima y el subíndice i enumera las distintas características del dato x.

Si se satisface la propiedad de independencia de las características, este algoritmo alcanza un resultado óptimo. Desafortunadamente, esto es bastante difícil de conseguir en la realidad, puesto que los atributos suelen guardar relación unos con otros [42].

2.3.2 k-Nearest neighbors (KNN)

Este procedimiento toma como premisa la asunción de que las muestras similares están unas cerca de otras. Se basa fundamentalmente en el cálculo de distancias entre puntos, utilizando normalmente la distancia euclídea (aunque dependiendo del problema pueden utilizarse otras como la chi-cuadrado). El algoritmo actúa de la siguiente forma [44]:

1. Se cargan los datos.
2. Se inicializa K al valor que se considere necesario. K es el número de vecinos, es decir, la cantidad de puntos más cercanos que se tendrán en cuenta para etiquetar las nuevas muestras.
3. Se calcula la distancia desde la nueva muestra al resto de ítems del set de entrenamiento.
4. Se seleccionan los K elementos más cercanos (con distancia más pequeña, según la función usada) y se etiqueta la nueva muestra con la etiqueta dominante entre estos K elementos seleccionados.

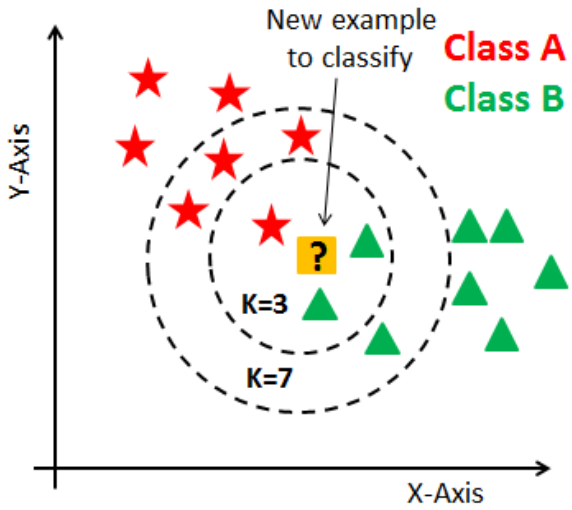


Figura 8: Ejemplo de clasificación del algoritmo KNN. Si K=3, el cuadrado amarillo pertenecerá a la clase B. Si K=7, el cuadrado amarillo pertenecerá a la clase A. Fuente: datacamp.com

Para hallar el valor adecuado de K se suele efectuar un barrido entre muchos valores hasta ver cuál es el que arroja mejores resultados. Al respecto de este tema resulta conveniente hacer un par de comentarios con respecto a los valores extremos de K:

- Si K=1, las predicciones tienden a volverse muy inestables. Si se tuviese el caso en el que los puntos de alrededor de la nueva muestra fuesen de la clase A menos el punto más cercano, perteneciente a la clase B, el algoritmo etiquetaría la nueva muestra como clase B, a pesar de que la gran mayoría de puntos colindantes son de la otra clase.

- Por el contrario, si se incrementa el valor de K se obtienen predicciones más estables debido a que se dispone de más puntos para verificar la etiqueta dominante entre los K vecinos. Sin embargo, si el valor de K es demasiado elevado se empiezan a tener fronteras poco precisas y, por ende, comienzan a aparecer errores.

2.3.3 Máquinas de soporte vectorial (SVM)

Las máquinas de soporte vectorial son muy versátiles y potentes, capaces de realizar clasificaciones lineales y no lineales, regresiones e incluso detección de valores atípicos en un conjunto de datos. La idea básica que se esconde detrás de este método es realizar la separación de instancias buscando el hiperplano que mejor divida el conjunto de datos disponible [42]. Como se muestra en la figura 3, el hiperplano que divide los dos conjuntos de características es la línea verde, realizándose así una separación lineal (las clases se pueden dividir mediante una recta).

Así pues, una vez que el método haya elegido el hiperplano óptimo, considerará que toda nueva instancia que quede a la izquierda de la línea verde pertenece a la clase negra, mientras que las nuevas instancias que queden a la derecha pertenecerán a la clase azul. Las muestras de cada clase que están más cerca del hiperplano se denominan vectores de soporte. El espacio existente entre los vectores de soporte y el plano separador se denomina margen [45].

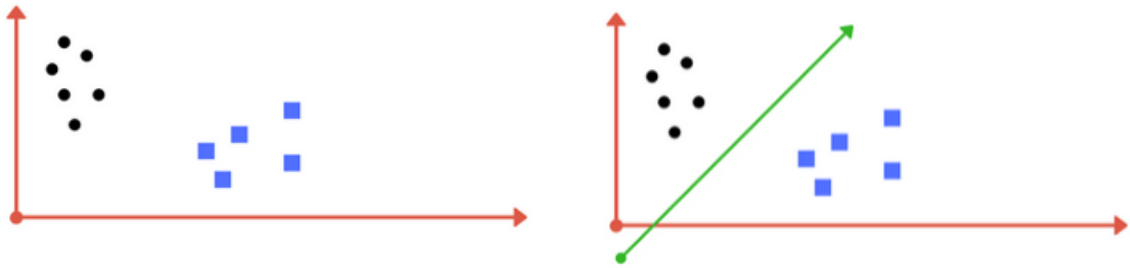


Figura 9: Ejemplo de la separación de clases realizada por una SVM. Fuente: medium.com

En ocasiones, no se puede recurrir a una separación lineal de variables para discernir entre las distintas clases, por lo que resulta necesario aplicar una transformación espacial que traslada los datos a un espacio donde el problema se convierte en lineal, permitiendo así la división. Una vez conseguido el hiperplano óptimo, se transforma la solución al espacio original. Estas transformaciones espaciales se conocen como kernels [46].

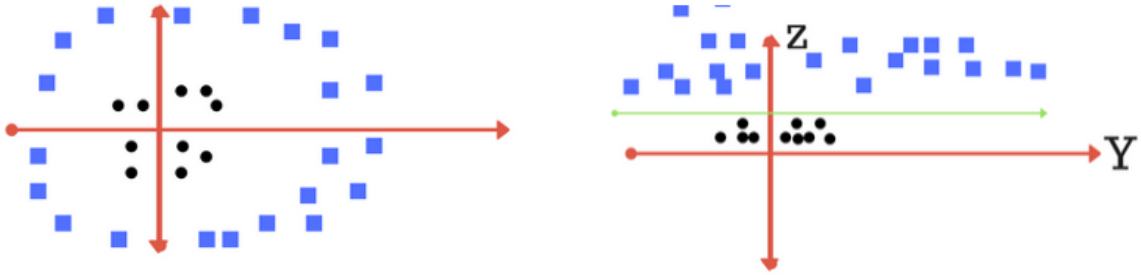


Figura 10: Ejemplo de utilización de una función kernel para separar las clases existentes. Fuente: medium.com

Por último, si las muestras de las clases están solapadas entre sí (las clases no son separables), podemos permitir cierta relajación a la hora de separar correctamente las clases permitiendo que haya muestras que estén mal clasificadas, o bien regular ciertos parámetros del método para hacer que el hiperplano se adapte mejor a estos puntos. Este proceso se denomina regulación de parámetros y los valores que pueden ser modificados se conocen como hiperparámetros. Los hiperparámetros más relevantes de una máquina de soporte vectorial son:

- C: también conocido como parámetro de regularización, controla el margen de separación con el hiperplano. Un valor de C pequeño permite un margen de separación más ancho, a pesar de que eso pueda llevar a clasificar algunas muestras incorrectamente; mientras que un valor grande de C hace que el margen se estreche, haciendo que se clasifiquen correctamente más puntos.

- Gamma: el parámetro gamma define el rango de puntos que serán considerados para calcular el hiperparámetro óptimo de separación de clases. Si el valor es grande, sólo se tendrán en cuenta los puntos más cercanos al posible hiperplano para calcular este último. Si por el contrario gamma es pequeño, también se tendrán en cuenta puntos más lejanos para efectuar el cálculo del hiperplano.

- Kernel: pueden utilizarse diferentes tipos de kernels en la SVM, como pueden ser el lineal, polinomial o exponencial, entre otros [45].

2.3.4 Árboles de decisión

Este método clasifica los datos gracias a una serie de reglas. La estructura del algoritmo resulta bastante visual al organizarse como un árbol, como se aprecia en la figura 11. En la cúspide de este esquema está el nodo raíz, que representa todas las muestras del set de datos, y que posteriormente se divide en dos o más ramas diferentes, subdividiéndose de esta forma el set de muestras. A los nodos que se dividen en dos o más subnodos se les conoce como nodos de decisión, y se van segregando las muestras más y más hasta llegar al nodo hoja, que representaría la clase con la que se etiquetarían las instancias que llegan hasta el final de esa rama del árbol.

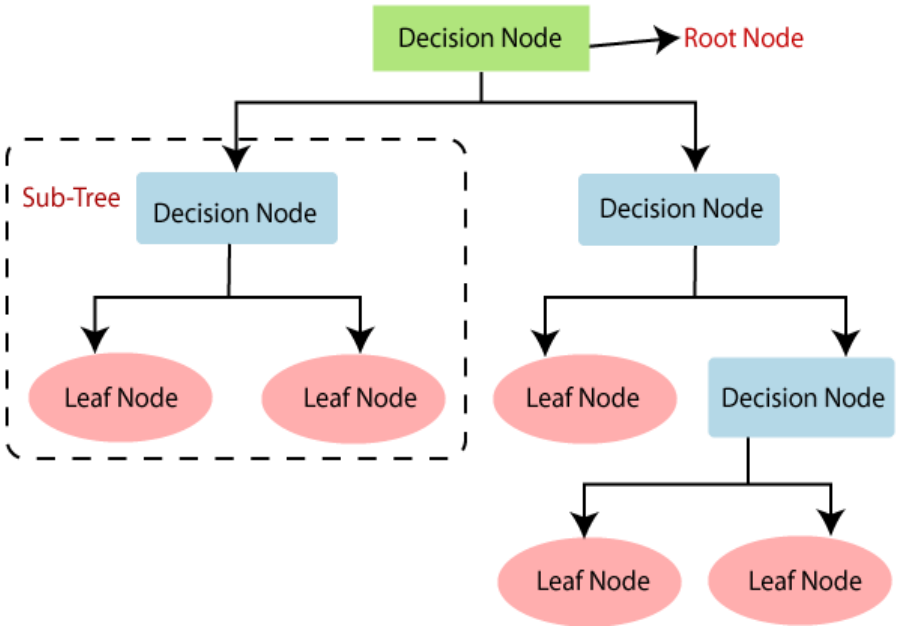


Figura 11: Estructura de un árbol de decisión. Fuente: javatpoint.com

Los árboles de decisión son uno de los métodos más extendidos en el aprendizaje máquina debido a su fácil interpretación, y al realizar de forma automática la exclusión de características irrelevantes en el proceso de construcción del árbol, entre otras muchas ventajas. Sin embargo, también tiene sus puntos débiles: uno de las más comunes es que este tipo de métodos tienden al sobreajuste si se construyen modelos muy complicados. Eso sí, este tipo de error puede corregirse gracias al ajuste de varios hiperparámetros, entre los que se cuentan los siguientes [47]:

- ***Mínimo número de muestras para una división (min_samples_split):*** Define el número mínimo de muestras que se requieren en un nodo para ser susceptible de ser dividido.

- ***Mínimo número de muestras para considerar un nodo hoja (min_samples_leaf):*** Define el número mínimo de muestras que deben existir en un nodo hoja o terminal. Si se tiene un problema de balanceo de clases se recomienda escoger un valor bajo.

- ***Profundidad del árbol (max_depth):*** Profundidad vertical (subniveles) que tiene el árbol.

- ***Número máximo de características (max_features):*** Número máximo de características consideradas a la hora de buscar la mejor división.

La precisión que arroja el algoritmo finalmente depende de las divisiones realizadas en los nodos. La creación de subnodos hace que cada vez éstos vayan siendo más homogéneos, es decir, todas las muestras que entran dentro de la división pertenecen a la misma clase. Dicho de otra forma, se puede decir que la pureza del nodo aumenta con respecto a la clase objetivo. El árbol de decisión divide los nodos en todas las características posibles y después selecciona la división que arroja mayor homogeneidad en los nodos. Los algoritmos de división más usados en los árboles de decisión para tareas de clasificación son:

I. Índice de Gini

Este algoritmo es usado principalmente por el algoritmo CART (uno de los varios métodos existentes para generar árboles de decisión). Trabaja con variables categóricas y realiza divisiones binarias, esto es, los nodos de decisión solamente tendrán dos subnodos, y así sucesivamente. El índice de Gini (G_i) puede oscilar entre 0 y 1, y se dice que un nodo es "puro" cuando su índice de Gini es 0 (todas las muestras pertenecen a la misma clase) [40]. $p_{i,k}$ representa la ratio de instancias de la clase k dentro de todas las instancias existentes en el i-ésimo nodo. n es el número total de clases existentes.

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \tag{2}$$

II. Chi-Square

Este procedimiento se emplea en el algoritmo CHAID de árboles de decisión. Trabaja con variables categóricas, y los nodos de decisión se pueden dividir en dos o más subnodos. Este indicador trabaja con variables objetivo binarias, y su finalidad es calcular la significación estadística de las diferencias existentes entre los nodos y sus subnodos correspondientes. El valor chi-cuadrado total se calcula sumando los valores chi-cuadrado de cada nodo, que se obtienen computando el cuadrado de la diferencia estandarizada entre las frecuencias observada y esperada de la variable objetivo [47]:

$$Chi - Square_{nodo} = \frac{(Frec.observada - Frec.esperada)^2}{Frec.esperada} \tag{3}$$

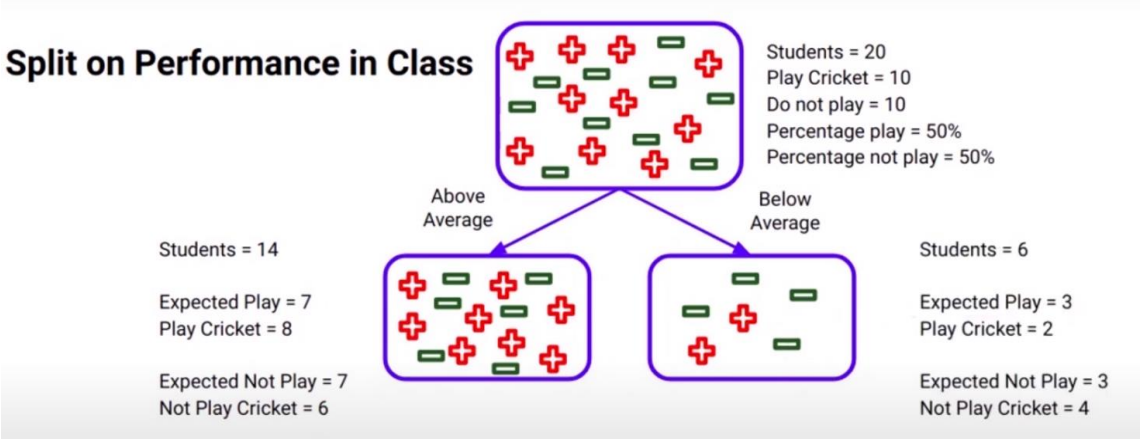


Figura 12: Ejemplo para calcular el valor chi-square. Fuente: analyticsvidhya.com

Para entender mejor el modus operandi de este método, se presenta el ejemplo de la figura 12. Se dispone de un set de 20 muestras que representan estudiantes, 10 de ellos juegan a cricket (Clase +, 50%) y los 10 restantes no (Clase -, 50%), y se realiza una división atendiendo a la variable objetivo que representa el desempeño de los estudiantes en la clase (por encima o por debajo de la media) [48].

Se calculan entonces los valores observados y esperados de esta división. En el nodo hijo "por encima de la media" tenemos 14 muestras, por lo que el valor esperado en este nodo de alumnos que juegan al cricket sería de 7 (dado que en el nodo padre existe un 50% de alumnos que juegan a cricket y un 50% que no). Sin embargo, vemos que en la realidad (valor observado) el número de alumnos que juegan a cricket en este nodo es de 8. Por tanto, el valor chi-cuadrado en este caso sería de:

$$Chi - Square_{juegan\ cricket,nodo\ izq} = \frac{(8-7)^2}{7} = 0.143 \tag{4}$$

Se realiza la misma operación en este nodo para los alumnos que no juegan cricket, y se calculan también los valores chi-square en el otro nodo hijo. El valor chi-cuadrado total es:

$$\begin{aligned} &Chi - Square_{juegan\ cricket,nodo\ izq} + Chi - Square_{NO\ juegan\ cricket,nodo\ izq} + \\ &+ Chi - Square_{juegan\ cricket,nodo\ der} + Chi - Square_{NO\ juegan\ cricket,nodo\ izq} \\ &= 0.143 + 0.143 + 0.3 + 0.3 = 0.946 \end{aligned} \tag{5}$$

Este proceso se repetiría para otras características del conjunto de instancias. Mientras más alto es el valor chi-cuadrado mayor es la significancia estadística entre el nodo padre y los nodos hijos, es decir, existirá más homogeneidad en los nodos hoja [48].

III. Ganancia de información

Utilizado principalmente por el algoritmo ID3. Para explicar este método, es necesario presentar dos términos bastante importantes en teoría de la información: la información y la entropía.

Matemáticamente, se define la información contenida en el valor a de esta forma:

Definición 1: *Dada una variable aleatoria discreta X que en un determinado experimento toma el valor* $a \in X$ *con probabilidad* $P_X(a)$*, decimos que la cantidad de información aportada por* a *o, simplemente, la información en* a *viene dada por* [49]:

$$\iota_X(a) = \log \frac{1}{P_X(a)} = -\log P_X(a), \qquad \forall a \in X \tag{6}$$

Por otro lado, la entropía viene descrita de la siguiente forma:

Definición 2: *Se define la entropía de la variable aleatoria discreta X, H(X), como el valor esperado de la variable aleatoria* $\iota_X(X)$ [49]*:*

$$H(X) = \mathbb{E}[\iota_X(X)] = \mathbb{E}\left[\log \frac{1}{P_X(X)}\right] = \sum_{a \in \chi} P_X(a) \log \frac{1}{P_X(a)} = \\ = \sum_{a \in \chi} P_X(a)\, \iota_X(a) \tag{7}$$

Dejando a un lado las definiciones formales, podría decirse que la información mide la incertidumbre de un evento en bits. Eventos que son más predecibles aportan menos información, mientras que eventos con mayor incertidumbre tienen más información. La entropía, por su parte, cuantifica cuánta información hay en una variable aleatoria. Este concepto es, precisamente, la clave de este procedimiento. La entropía puede servir para computar la pureza de los distintos nodos del árbol de decisión, ya que si este valor es 0 significa que todas las instancias son idénticas, es decir, que todas las instancias son de la misma clase. Precisamente la reducción de la entropía es lo que va buscando este método; esta idea se conoce como ganancia de información [50].

Para calcular la entropía de cada nodo hijo, en primer lugar calculamos la entropía del nodo padre teniendo en cuenta todo el set de datos *S* y seguidamente, la de los nodos hijos. Después, se calcula la entropía condicional del conjunto de datos dada la característica *a*: se divide el conjunto de instancias en grupos para cada uno de los distintos valores v_i que toma la característica *a*; y se calcula la suma del ratio de instancias existentes en el dataset en los que la variable *a* toma el valor v_i multiplicado por la entropía de cada grupo [50]. Finalmente, la ganancia de información se obtiene restando estas dos cantidades:

$$IG(S, a) = H(S) - H(S|a) = \\ = H(S) - \sum_{i}^{v} \frac{n^{\underline{o}}\ instancias\ v_i}{S} * H(a, v_i) \tag{8}$$

2.3.5 Red neuronal profunda (DNN)

Para comprender en su totalidad las redes profundas, es necesario hablar (aunque sea de forma breve) del Perceptrón, la primera red neuronal básica que se desarolló a mediados de los 50 y que está inspirado en el comportamiento de una neurona humana.

La Figura 13 representa la estructura de un perceptrón. Está compuesto por un conjunto de entradas, cada una asociada a un peso. Este peso se ajusta de forma automática a medida que la red neuronal va realizando el proceso de aprendizaje. Posteriormente, se realiza el producto escalar de cada una de las entradas por su peso, y se suman todas las operaciones. El resultado de esta suma resulta ser la entrada de la función de activación (para el Perceptrón la función de activación es una función escalón, pero posteriormente se introdujeron otro tipo de funciones de activación distintas). La función de activación se encarga de mantener el conjunto de valores de salida dentro de un rango determinado (para la función escalón, en el rango [0,1]) y devuelve este valor como valor de salida [51].

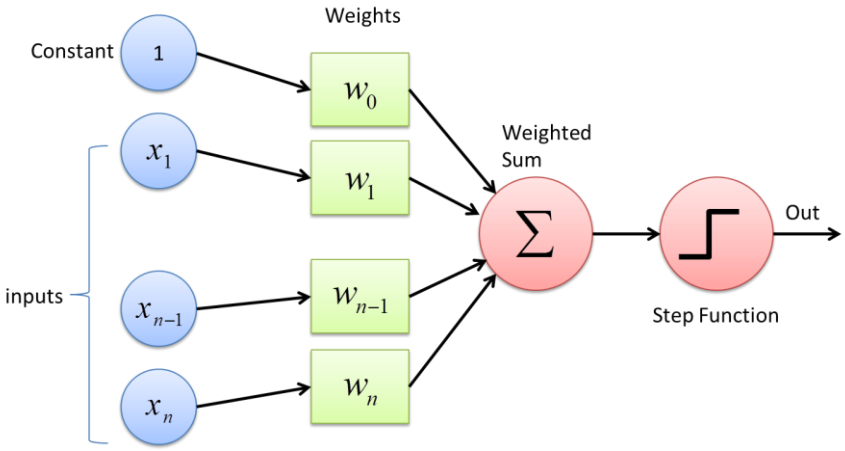


Figura 13: Estructura de un perceptrón. Fuente: deepAI.com

Esta estructura fue el punto de partida de las neuronas artificiales, las cuales conforman la estructura de los modelos de aprendizaje profundo. Las redes neuronales artificiales constan de una capa de entrada, una o muchas capas ocultas, completamente conectadas entre sí (dependiendo del número de capas ocultas se tratará de una red neuronal simple o profunda) y una capa de salida.

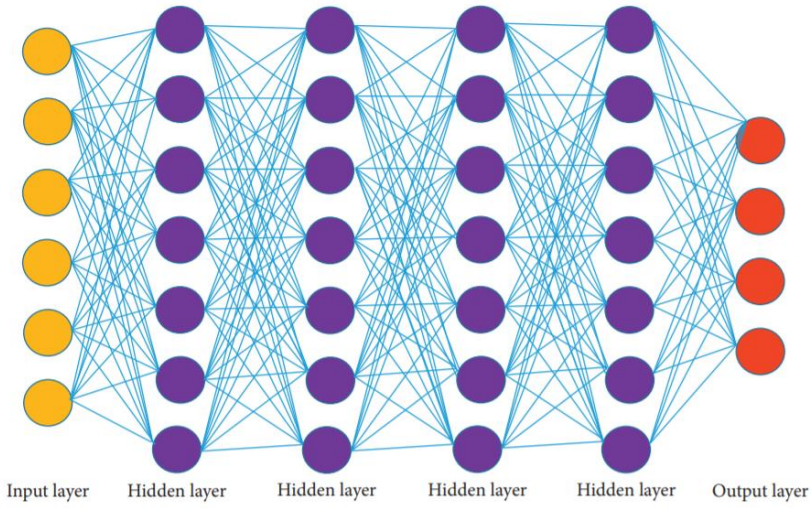


Figura 14: Estructura simplificada de una red neuronal profunda. Cada círculo representa una neurona. Fuente: A Review of Deep Learning Security and Privacy Defensive Techniques. Tariq, I. (2020)

En la capa de entrada se introducen los datos que servirán para alimentar a la red. Normalmente, el número de neuronas existentes en esta capa es el mismo que el número de características de entrada de los datos. Esta capa está seguida de una o varias capas ocultas, y, dependiendo de la arquitectura, pueden estar conectadas completamente o no a la capa oculta que les sigue.

A la capa de entrada le sigue una sucesión de capas ocultas. Estas capas manejan unos parámetros denominados pesos, los cuales determinan el aprendizaje de la red. Gracias a los algoritmos de asignación y actualización de pesos (como el SGD o el Momentum), se encuentra la combinación óptima de pesos que permite dar la mejor solución a la red, minimizando el error de la función de pérdida (función que mide el error entre la salida obtenida y la deseada) [51]. La búsqueda de estos pesos es un proceso iterativo, y la actualización de ellos se realiza a través de la propagación de la información a lo largo de la red. Existen dos métodos posibles: se puede realizar *backpropagation* o propagación hacia atrás, en la que se propagan los errores medidos en la capa de salida hasta la capa de entrada para actualizar los parámetros, o bien se puede realizar *forward propagation* o propagación hacia adelante, transmitiéndose así los parámetros de la capa de entrada hasta la capa de salida [52].

Por último, la capa de salida proporciona la predicción generada por el modelo. Dependiendo de la configuración de la red, la salida final puede ser un valor concreto (regresión) o un conjunto de probabilidades (clasificación). Esto está controlado por el tipo de función de activación utilizado en las neuronas de la capa de salida [51].

2.3.6 Red neuronal convolucional (CNN)

Este tipo de algoritmo de aprendizaje profundo cosecha resultados excelentes en aplicaciones relativas a imágenes y en el campo de la visión computacional en general. Una red neuronal convolucional consta de una sucesión de capas que van realizando sucesivamente operaciones en los datos de entrada. Como se observa en la figura 2, la estructura de una CNN es una sucesión de capas convolucionales y capas de pooling, terminando en una capa completamente conectada que es la que proporciona la salida del modelo. De forma breve, se podría decir que las capas convolucionales se encargan de extraer características de la imagen introducida como entrada, y las capas de pooling generalizan los atributos extraídos [42].

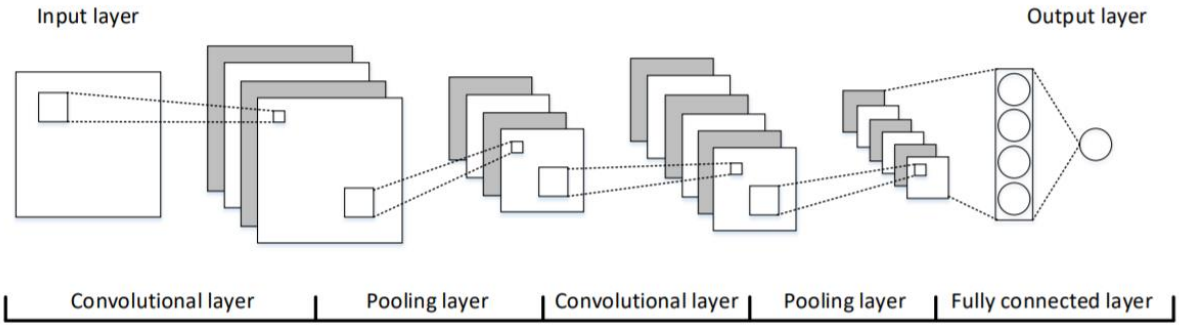


Figura 15: Estructura de una CNN. Fuente: [42]

Entrando un poco más en detalle sobre cómo funciona este modelo, primeramente la red recibe una imagen compuesta de píxeles, cuyos valores se suelen normalizar antes de introducir la matriz como entrada de la red (se supone que la imagen está en escala de grises para simplificar la explicación)[2] . Esta imagen de mxn píxeles llega a la capa convolucional, donde se aplican las llamadas "convoluciones": se coge un grupo de píxeles cercanos de la imagen de entrada y se aplica un producto escalar de este grupo de píxeles contra una matriz llamada kernel o filtro. El kernel va recorriendo toda la imagen, realizando el producto matricial con el grupo de píxeles correspondiente y desplazándose un píxel de izquierda a derecha y de arriba abajo hasta recorrer toda la imagen, generando así una matriz de salida como se observa en la figura 41. En general, no se aplica solamente un filtro a la imagen, sino un conjunto de ellos generando cada uno su respectiva matriz de salida, la cual destaca una característica de la imagen. Estas matrices de salida hacen que el número de neuronas de las capas se haga muy elevado (por ejemplo, si se supone que la imagen es de 20x20 píxeles y se aplican 30 filtros, el número de neuronas de esta capa sería de 20*20*30 = 12.000 neuronas) [53].

Aquí es donde entran en acción las capas de pooling, encargadas de reducir la dimensionalidad de las salidas producidas por las capas convolucionales a través de diversas técnicas de submuestreo; así se evita que el crecimiento del número de neuronas de capa a capa crezca de forma descontrolada. Las técnicas de submuestreo empleadas reducen el tamaño de las imágenes de salida de cada capa convolucional, pero conservando las características más importantes que detectaron los filtros en la imagen.

Posteriormente, se sigue esta sucesión de capas convolucionales y capas de pooling, siendo las primeras cada vez más capaces de reconocer características más complejas a medida que la imagen va pasando capas. Finalmente, se llega a la capa completamente conectada, que se encarga de extraer las características generales de lo que recibe como entrada y clasificar la imagen pertinentemente [53].

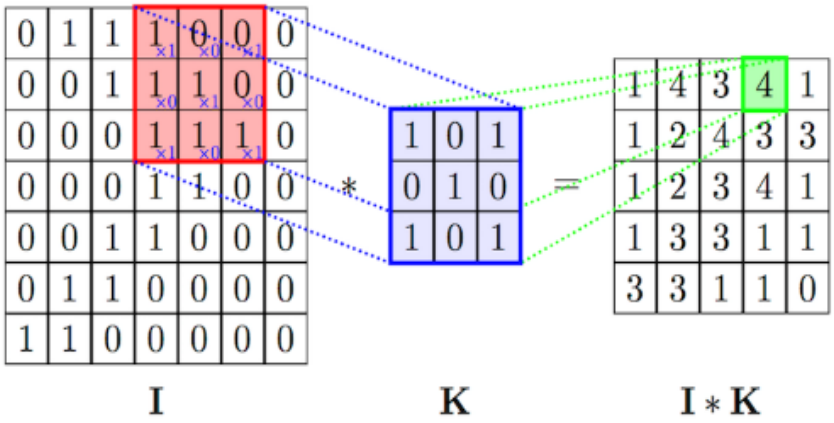


*Figura 16: Operación de convolución realizada en las capas de convolución. I es la imagen que se introduce como entrada en la red, K es el kernel e I*K es la salida de la capa convolucional, resultado de efectuar la operación de convolución.*

[2] *La extensión de la explicación dada a imágenes a color es inmediata: solamente habría que aplicar el filtro a cada una de las 3 capas de color (R, G, B), y sumar los 3 filtros que conformarían una única salida.*

2.3.7 Red neuronal recurrente (RNN)

Las RNN se usan para trabajar principalmente con datos secuencias y son ampliamente usadas en tareas relacionadas con el procesamiento del lenguaje natural. Analizar datos secuenciales de forma individual no tiene sentido; es por eso que cada neurona de una RNN no recibe únicamente el estado actual sino también los estados anteriores. Esta es la única diferencia estructural con respecto a una red neuronal común. Debido a la estructura que tiene esta red, las RNN sufren a menudo de problemas de "vanishing gradients" o "exploding gradients". En pos de resolver esta dependencia a largo plazo de las entradas, se han propuesto variantes para solucionarlo como la memoria a largo plazo o LSTM [42].

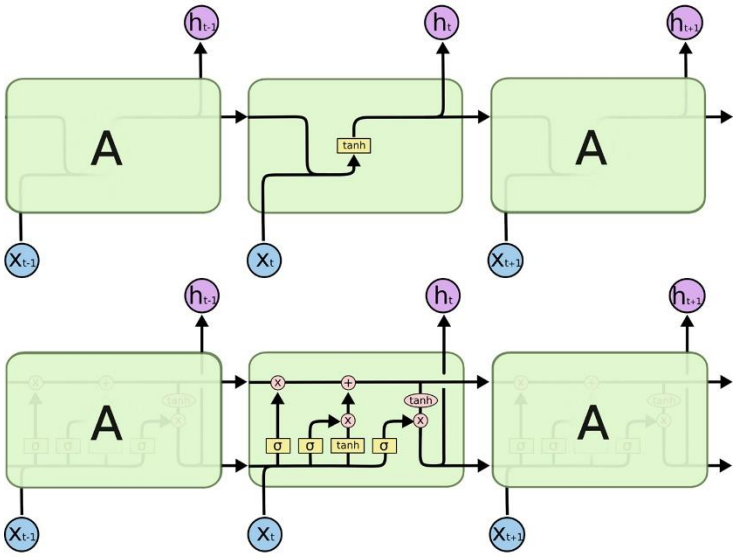


Figura 17: Diferencia entre una celda de una RNN (figura de arriba) y una celda LSTM (figura de abajo). Fuente: "Understanding LSTM Networks", de Christopher Olah.

Cada unidad o celda de LSTM contiene tres puertas: una puerta de olvido, una puerta de entrada y otra de salida. La primera de ellas elimina memoria que ha quedado obsoleta; la puerta de entrada es la encargada de recepcionar nuevos datos y la puerta de salida se encarga de combinar las memorias a largo y corto plazo para generar el estado actual de memoria del modelo.

2.3.8 Máquina de Boltzmann restringida (RBM)

Las máquinas de Boltzmann están basadas en neuronas estocásticas: en lugar de usar una función de activación determinista (como la función escalón) para concretar el valor de salida, las neuronas computan un valor de 1 con una probabilidad, o 0 en otro caso. La función de probabilidad usada por esta red está basada, obviamente, en la distribución de Boltzmann.

Debido a su naturaleza estocástica, este tipo de red nunca llegará a estabilizarse en torno a una configuración fija, si no que irá permutando entre distintas configuraciones. Si la red funciona durante una cantidad suficiente de tiempo, la configuración adoptada por ella solamente será función de los pesos de las neuronas y los términos de sesgo, y no de la configuración inicial de la red. Cuando la red alcanza este estado se dice que está en "equilibrio térmico" y, una vez llegada a este punto, es capaz de simular una amplia variedad de distribuciones de probabilidad [40]. Por

este motivo las máquinas de Boltzmann también se encuadran dentro de las llamadas redes generativas profundas, modelos capaces de generar muestras sintéticas a partir de lo que han aprendido. Sin embargo, no existen formas eficientes de entrenamiento de estas redes, problema que se soluciona con los algoritmos para entrenar máquinas de Boltzmann restringidas.

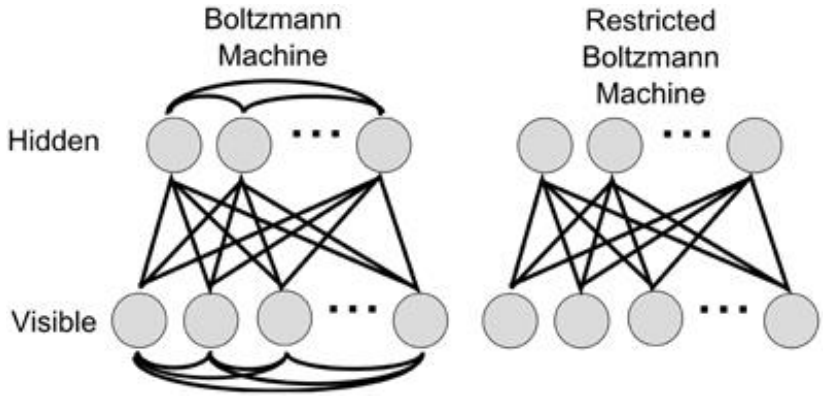


Figura 18: Estructuras de una máquina de Boltzmann genérica y una máquina de Boltzmann restringida. Fuente: Real-Time Classification and Sensor Fusion with a Spiking Deep Belief Network. O'Connor, P. et al. (2013)

Una máquina de Boltzmann restringida, a grandes rasgos, se comporta de la misma forma que una máquina de Boltzmann genérica, con la salvedad de que no existen conexiones entre las neuronas de la capa visible o entre las neuronas de la capa oculta. Además, esta red neuronal se entrena con un algoritmo denominado divergencia contrastiva, introducido en 2005 por los investigadores Miguel Ángel Carreira-Perpiñán y Geoffrey Hinton[3]. Este algoritmo permite entrenar este tipo de redes de forma mucho más eficiente que otros que se venían usando, propiciando el éxito de los modelos basados en múltiples redes de Boltzmann apiladas. Este tipo de redes se pueden usar de muchas formas distintas, como pueden ser la corrección de imágenes incompletas o que presenten ruido, o bien para tareas de clasificación de muestras.

2.3.9 Autocodificadores

Los autocodificadores surgieron en la década de 1980 de la mano de Geoffrey Hinton y David Rumelhart. Son redes neuronales artificiales capaces de aprender representaciones de los datos de entrada, llamadas codificaciones, sin ninguna supervisión (es decir, sin que las muestras estén etiquetadas). El funcionamiento a grandes rasgos de estos modelos reside en aprender a través del copiado de las entradas en las salidas. A priori puede parecer una tarea sencilla, pero diversas restricciones con las que se puede encontrar el modelo complican esta acción. Por ejemplo, se puede introducir ruido a las muestras de entrada y entrenar la red para que ésta recupere las entradas originales. Esto hace que la copia de las entradas en las salidas no sea ya una tarea trivial, y así se obliga al autocodificador a encontrar formas más eficientes de representar los datos.

Debido a que las codificaciones suelen estar representadas en un espacio de dimensiones menor que el espacio de dimensiones de los datos de entrada, los autocodificadores resultan bastante potentes en cuanto a reducción de dimensionalidad (superando las prestaciones del PCA, restringido solamente a la realización de transformaciones lineales), así como también se usan

[3] *Geoffrey Hinton también fue el creador de la máquina de Boltzmann en 1985, junto con Terrence Sejnowski.*

para seleccionar las características más importantes de un conjunto de datos, generación de muestras sintéticas o clasificación [40].

De todos los modelos existentes en el grupo de los autocodificadores, en este trabajo se centra la atención en los autocodificadores variacionales (VAE), modelo que será tomado como base para el desarrollo de un clasificador de ataques a una red de telecomunicaciones. En un capítulo posterior se desarrollará de forma más extensa y particularizada los detalles y funcionamiento de este algoritmo de la familia de los autocodificadores.

2.4 Flujo de trabajo del diseño de un algoritmo de ML

El desarrollo de un algoritmo de aprendizaje máquina requiere de varios pasos a realizar hasta conseguir un código completamente testado y en condiciones de estar operativo para las tareas para las que ha sido diseñado. En este apartado se recoge precisamente una hoja de ruta a seguir para realizar un algoritmo de ML, describiendo pormenorizadamente cada uno de los pasos a ejecutar.

Resulta importante poner de manifiesto que el flujo de trabajo presentado recoge todas las tareas que se podrían realizar a la hora de diseñar un algoritmo de estas características, y en la realidad puede ser que no resulte obligatorio tener que cubrir algunas de ellas; bien porque el problema no lo requiere o bien porque simplemente no se considera dicha tarea en el desarrollo del método. En resumidas cuentas, el procedimiento de creación de un algoritmo de ML no es una ciencia exacta y algunos pasos pueden ser omitidos en el proceso, o bien no ser ejecutados en el mismo orden en el que se exponen en las siguientes líneas.

2.4.1 Obtención de datos y visualización de su estructura

La obtención de datos es un paso fundamental, ya que gran parte del éxito que tenga el método que se está desarrollando pasa por recabar una cantidad de datos ingente y de calidad. Existen varias formas de conseguir datos para el algoritmo: la más fácil pasa por hacerse con una base de datos ya existente, aunque en ocasiones, muchas de estas bases de datos no son de dominio público. A pesar de ello, existen bancos de datos suficientes como para entrenar el método, sea el problema que sea. Otra opción es aprovechar la cantidad de información disponible en internet y crear desde cero una base de datos, recolectando información automáticamente a través de diversas webs. Esta técnica se conoce como web scrapping [54].

Una vez se tienen los datos recopilados, es bastante recomendable visualizar los datos para encontrar si existen ciertos patrones a los que prestar atención, o si los datos siguen algún tipo de distribución. También podremos descubrir si existen características correladas entre sí (en ese caso, podríamos eliminar una de ellas), si el conjunto de datos está imbalanceado (no existe equidad de muestras entre las distintas clases) o si existen "outliers" que desvían de forma importante las distribuciones (un "outlier" es aquel valor que difiere significativamente del resto de muestras). Esta etapa de previsualización de la información puede arrojar unas primeras conclusiones sobre el conjunto de datos que será preprocesado en el siguiente paso [54].

Como se comentó en líneas anteriores, una de las técnicas más usuales para visualizar los datos es el t-SNE o técnica de incrustación de vecinos estocástica distribuida. Debido a su uso en posteriores capítulos de este trabajo, se procede a dar una explicación más extensa del funcionamiento de este método.

I. Técnica de incrustación de vecinos estocástica distribuida (t-SNE).

La t-SNE es una técnica de reducción de dimensionalidad utilizada especialmente en bases de datos con gran número de características. En resumidas cuentas, este algoritmo intenta minimizar la divergencia entre dos distribuciones de probabilidad: la primera es una distribución de probabilidad sobre pares de muestras en el espacio original que mide la similitud entre dichos pares y la segunda hace lo propio con los pares de muestras, pero en un espacio de menores dimensiones. Entrando en más detalle, la técnica realiza los siguientes pasos [55]:

1. Se tiene un set de datos de alta dimensionalidad $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_N\}$ (donde N es el número total de puntos) y una función $d = (\boldsymbol{x}_i, \boldsymbol{x}_j)$ que calcula la distancia entre un par de instancias (por ejemplo, la distancia Euclídea). El objetivo es conseguir un espacio s-dimensional donde cada instancia esté representada por un punto, $\mathcal{E} = \{\boldsymbol{y}_1, \boldsymbol{y}_2 \dots \boldsymbol{y}_N\}$, con $\boldsymbol{y}_i \in \mathbb{R}^s$, siendo valores típicos para s 2 ó 3.
2. Para la primera distribución de probabilidad en el espacio original, se define la probabilidad conjunta

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{9}$$

que mide la similitud entre las instancias $\boldsymbol{x}_i$ y $\boldsymbol{x}_j$. Según los creadores de este algoritmo, la similitud de $\boldsymbol{x}_i$ con respecto a $\boldsymbol{x}_j$ se define como "la probabilidad condicional de que $\boldsymbol{x}_i$ escogiese a $\boldsymbol{x}_j$ como su vecino si los vecinos fuesen escogidos de forma proporcional a su densidad de probabilidad bajo una gaussiana centrada en $\boldsymbol{x}_i$" . Matemáticamente, esta probabilidad condicional $p_{j|i}$ se define de la siguiente forma [56]:

$$p_{j|i} = \frac{\exp(-\frac{d(x_i, x_j)^2}{2\sigma_i^2})}{\sum_{k \neq i} \exp(-\frac{d(x_i, x_k)^2}{2\sigma_i^2})} \qquad p_{i|i} = 0. \tag{10}$$

donde σ_i es la varianza de la gaussiana centrada en $\boldsymbol{x}_i$.

Dicho de otra forma, en este primer paso se centra en cada uno de los puntos $\boldsymbol{x}_i$ una distribución gaussiana para definir las relaciones entre los puntos en el espacio de características original. Para puntos que están lejos de $\boldsymbol{x}_i$, la probabilidad de ser escogido como vecino es muy pequeña [57].

3. Para la distribución de probabilidad en el espacio de menores dimensiones, Lo que se busca es que dicha distribución sea lo más parecida posible a la del espacio original. Intuitivamente, se podría pensar que lo mejor sería adoptar una distribución gaussiana también en este caso, pero las colas de dicha distribución son cortas y esto hace que los puntos tiendan a agolparse en este nuevo espacio. Para distribuir de mejor forma dichos puntos, t-SNE utiliza una distribución t-Student, la cual presenta una caída rápida y colas más largas [57]. Se computa pues, la siguiente probabilidad conjunta:

$$q_{j|i} = \frac{\left(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2\right)^{-1}}{\sum_{k \neq l} (1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|^2)^{-1}} \tag{11}$$

siendo $\boldsymbol{y}_i$ los puntos del espacio de características original, pero mapeados ahora en el espacio de dimensiones reducidas.

4. Por último, la localización de los puntos $\boldsymbol{y}_i$ se determina minimizando la divergencia Kullback-Leibler (divergencia KL) entre las dos distribuciones de probabilidad. Típicamente se minimiza descendiendo a lo largo del gradiente [55]:

$$C(\mathcal{E}) = KL(P||Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$
$$\frac{\partial C}{\partial \boldsymbol{y}_i} = 4 \sum_{i \neq j} (p_{ij} - q_{ij}) q_{ij} Z(\boldsymbol{y}_i - \boldsymbol{y}_j) \quad (12)$$

siendo $Z = \sum_{k \neq l} \left(1 + \left\|\boldsymbol{y}_k - \boldsymbol{y}_l\right\|^2\right)^{-1}$ un término de normalización.

2.4.2 Preprocesado de datos

El preprocesado del set de instancias es fundamental por muchos motivos. En primer lugar, un correcto tratamiento de los datos normalmente mejora los resultados obtenidos, y, además, aligera sustancialmente el tiempo de ejecución a la hora de entrenar el método. Los datos se pueden procesar de muchas maneras y, en este caso, no es obligatorio llevar a cabo todas las opciones mostradas a continuación. Dependerá del conjunto de datos que se maneje y de la propia decisión del creador del algoritmo, ya que puede ser que no se obtenga ningún tipo de ventaja al realizar cierto tratamiento de los datos. Entre los distintos procedimientos a realizar se cuentan:

- ***Tratamiento de valores duplicados y nulos:*** Los métodos de aprendizaje máquina funcionan mejor si se eliminan las instancias duplicadas. Si no se hace podría llevar al método a comportarse erróneamente. En cuanto al tratamiento de valores nulos se puede optar por diversas opciones: la primera de ellas es eliminar estas instancias de la misma forma que se hace con los valores duplicados. No obstante, aunque resulta una solución rápida y cómoda, sólo funcionaría sin afectar a los datos si la proporción de valores a eliminar resulta bastante baja en comparación con el dataset completo (menos de un 10% de los datos). Si el número de valores nulos empieza a ser comparable es necesario elegir otras opciones, como completar los valores con la media, la mediana o la moda de la característica a la que pertenecen los valores nulos (para valores nulos numéricos) o reemplazarlos con el valor más frecuente (para valores nulos categóricos) [58].

- ***Balanceo de clases:*** El problema de las clases imbalanceadas consiste en tener mucha representación de muestras de algunas clases y muy poca de otras. Trabajar con clases imbalanceadas puede hacer que el algoritmo de aprendizaje máquina realice un aprendizaje sesgado, esto es, aprenderá correctamente cómo clasificar las muestras mayoritarias y no podrá hacer lo propio con las muestras de las clases con menor representación. Lo recomendable en primera instancia es realizar el entrenamiento y evaluación del algoritmo con el conjunto de datos original; si éste generaliza correctamente no hay que realizar nada más. Si no es el caso, se debe aplicar una técnica de balanceo de clases [59]. Se puede optar por aplicar "undersampling", que consiste en eliminar muestras aleatorias de las clases mayoritarias, o bien la técnica contraria ("oversampling"), que genera muestras sintéticas de las clases con menor representación [60]. Otra opción es la de aplicar pesos a la función de coste del modelo en cuestión, penalizando las clases mayoritarias y dando mayor peso a las clases más raras.

- ***Normalización y codificación:*** La normalización es una técnica que cambia los valores de las columnas numéricas para usar una escala común en todos ellos, sin distorsionar las diferencias existentes entre los rangos de valores ni perder información. Algunos algoritmos necesitan de este procedimiento para poder procesar los datos correctamente. Las técnicas de normalización más usadas son la estándar (se sustrae la media y se escalan los datos a varianza 1) y la MinMax (se asigna al valor más pequeño de la característica el 0 y al mayor el 1; el resto estarán entre ese rango de valores, pertinentemente escalados) [61]. Para variables categóricas, la estrategia más común que se adopta es la codificación *one-hot*, en la que se crea una columna para cada valor distinto que existe en la característica que se está codificando y, para cada instancia de la base de datos, se rellena con un 1 la columna a la que pertenece dicho registro, dejando las demás a 0 [62].

- ***Selección de características o reducción de dimensiones:*** En principio, construir un modelo de aprendizaje máquina no requiere selección de características. Sin embargo, cuando el número de características empieza a hacerse muy elevado en comparación con el número de muestras total del conjunto es recomendable hacerlo. El no emplear selección de características cuando se está en esta tesitura puede guiar al modelo a sufrir sobreajuste; por eso se balancea el número de características en proporción al número de instancias existente. Los métodos de selección de características se suelen dividir en 3 métodos principalmente [63]:

    - Métodos de filtrado: realizan la selección de "features" en dos pasos: primero el método evalúa la capacidad de cada característica de discernir entre las distintas clases y seguidamente, se eliminan las características que superan un cierto umbral, quedando así un menor número de atributos.

    - Métodos "*wrapper*": buscan el mejor subconjunto de características a través de algoritmos de clasificación. El objetivo es dar con el subset de atributos que mejor discriminan las clases.

    - Métodos integrados: estos métodos realizan la selección de características durante el entrenamiento del algoritmo, realizándose al mismo tiempo el proceso de clasificación y el proceso de selección de características [64].

  En el caso de la reducción de dimensionalidad, el objetivo es combinar las características existentes para crear un espacio de características de menor dimensión que el original y que permita discernir entre las clases apropiadamente [63]. Entre estos algoritmos se encuentra el análisis de componentes principales o PCA.

- ***Separación de datos en entrenamiento y test:*** por último, es necesario separar el set de datos en dos subsets: uno dedicado al entrenamiento del modelo y otro a la evaluación. El porcentaje dedicado a cada tarea no es fijo; se suele fraccionar en torno a un 80/20, 70/30, etc., dependiendo del caso y del volumen de datos existente. Para mejorar el modelo y ajustar sus parámetros se puede aplicar la técnica de validación cruzada, donde en lugar de dos conjuntos de datos se tienen tres: entrenamiento, test y validación. Este último conjunto forma parte del conjunto de entrenamiento, y se utiliza durante iteraciones que se harán con el conjunto de entrenamiento, utilizándose el conjunto de test solamente para evaluar la solución obtenida al ajustar pertinentemente los parámetros del modelo [65].

La validación cruzada más usada es la *K-fold*, sobre todo si se sufre de no tener suficientes muestras en el set de datos. En esta técnica se divide el set de datos en K grupos, de los que K-1 se emplean para entrenar el modelo y el restante, para evaluarlo. Posteriormente, se itera de nuevo, cambiando el subset utilizado para evaluar el método.

Otra técnica de validación cruzada es la división entrenamiento-test estratificada, utilizada para atajar el problema de clases imbalanceadas. En ella, se realizan las divisiones de los distintos subsets de datos respetando la proporción original de instancias de cada clase en el conjunto total [66].

2.4.3 Elección del modelo, entrenamiento y evaluación

La elección del método deberá de ser adecuada al tipo de problema que se quiera tratar. En anteriores apartados se ha presentado una revisión amplia de algoritmos relacionados con problemas de clasificación al ser el problema que se trata en este documento, pero la casuística de tareas es mucho más amplia: procesamiento de texto, imágenes, regresión, clustering, predicción, etc.

Una vez decidido el modelo adecuado para dar solución al problema que se tiene entre manos, el siguiente paso es entrenar la máquina con el subset de datos de entrenamiento configurando correctamente los hiperparámetros del modelo y, posteriormente, evaluarla con el set de testeo. Esto servirá para verificar la precisión del algoritmo (entre otras métricas) y determinar la confianza que inspira el modelo [54].

2.4.4 Configuración de hiperparámetros

Si los resultados obtenidos al finalizar la evaluación del modelo no son satisfactorios puede ser que el modelo haya sufrido sobreajuste (el modelo se ha ceñido demasiado a los datos de entrenamiento y no generaliza correctamente los datos de test) o subajuste (el modelo no clasifica correctamente ninguna instancia porque es muy simple). Para corregir estos problemas se recurre habitualmente a cambiar la configuración de los hiperparámetros del algoritmo. Se puede incrementar la cantidad de iteraciones de los datos de entrenamiento (epochs), o la tasa de aprendizaje o "learning rate", hiperparámetro que multiplica al gradiente para acercarlo poco a poco al mínimo global y minimizar el coste de la función. Este proceso no es una ciencia exacta y la única manera de dar con un ajuste de hiperparámetros óptimo es probando con distintas combinaciones de valores [54].

2.4.5 Lanzamiento y monitorización del sistema

El último paso de todo este proceso pasa por llevar el algoritmo a un entorno de producción. Resulta conveniente escribir código que permita monitorizar el sistema en tiempo real, y que sea capaz de lanzar alertas si tiene algún fallo. Así pues, no solo se previenen las caídas inesperadas del sistema, sino que también se percibe la degradación del rendimiento del sistema. Esto es bastante habitual en la realidad, y para solventarlo es necesario reentrenar los modelos con nuevas instancias. También resulta conveniente monitorizar la entrada de datos al algoritmo, ya que el rendimiento de éste se puede ver severamente afectado si los datos con los que se alimenta al método no son de calidad [40].

2.5 Métricas para evaluar el rendimiento del algoritmo

Las métricas empleadas para la evaluación de los algoritmos de aprendizaje máquina son diversas y también dependen del problema que se esté atacando. En este apartado se destacarán en concreto las medidas más comunes empleadas para evaluar la actuación de clasificadores; dichas métricas serán empleadas en capítulos posteriores para medir el desempeño del algoritmo de aprendizaje máquina desarrollado.

I. Matriz de confusión

La matriz de confusión no se corresponde con una métrica propiamente dicha, pero resulta una forma muy visual de apreciar si el algoritmo desarrollado clasifica correctamente o no. Para computar una matriz de confusión, es necesario tener tanto las predicciones realizadas por el sistema como las etiquetas verdaderas correspondientes a cada instancia. De esta forma, se puede contabilizar el número de veces que una instancia es clasificada con la etiqueta correcta por parte del sistema o no. La figura 7 muestra el esquema de una matriz de confusión perteneciente a una clasificación multiclase.

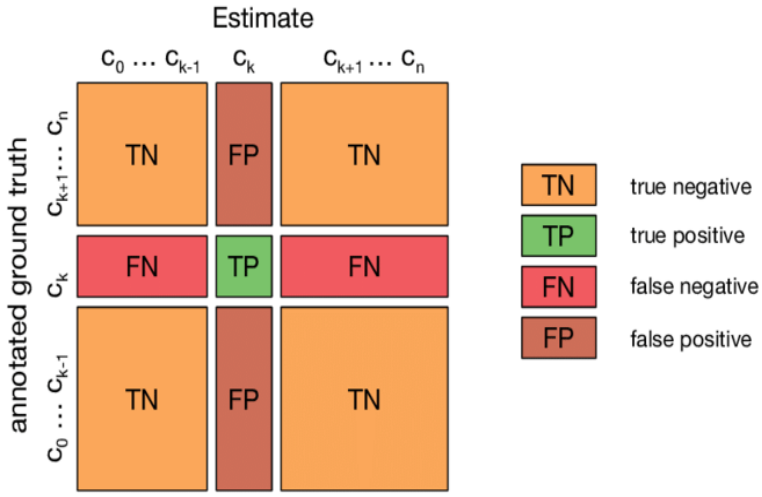


Figura 19: Estructura de una matriz de confusión para una clasificación multiclase. Fuente: "Activity, Context, and Plan Recognition with Computational Causal Behaviour Mode" – Frank Krüger

Cada fila de la matriz de confusión representa la clase verdadera de la muestra, mientras que las columnas representan las clases predichas por el método. Se pueden extraer 4 valores distintos de la matriz de confusión, atendiendo a la correcta clasificación o no de las muestras por parte del algoritmo. Centrándonos en la fila C_k, si las instancias etiquetadas por el algoritmo coinciden con la clase verdadera de dicha instancia (k), se habla de verdaderos positivos (TP). Si, por el contrario, el algoritmo etiqueta instancias pertenecientes a la clase k con otras etiquetas, se tienen falsos negativos (FN). Los falsos positivos (FP) se corresponden con aquellas muestras etiquetadas con la clase k por el algoritmo, pero que pertenecen a otras clases distintas, mientras que los verdaderos negativos (TN) se corresponden con las muestras que pertenecen a otras clases y que no han sido etiquetadas por el algoritmo como pertenecientes a la clase k.

La matriz de confusión arroja mucha información con sólo echar un vistazo, pero en ocasiones es preferible una métrica más concreta para evaluar los resultados, como pueden ser las que siguen [40].

II. Precisión (Precision)

La precisión mide qué porcentaje de clasificaciones positivas ha sido correcta, y viene dada por la siguiente fórmula:

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

Es conveniente acompañar esta métrica de otras como la exhaustividad, ya que en ocasiones altos porcentajes de precisión no se corresponden con un buen desempeño del algoritmo. Por ejemplo, si se tiene una distribución imbalanceada de clases en las que se tiene 1000 muestras de una categoría 1 y 100 sólo de la categoría 2. Si el modelo clasifica todas las muestras como pertenecientes a la categoría 1, la precisión obtenida sería del 90.9% cuando realmente, el modelo solamente predice una clase correctamente.

III. Exhaustividad o Sensibilidad (Recall)

La exhaustividad (o tasa de verdaderos positivos) se define como la ratio de instancias positivas detectadas correctamente por el clasificador [40]:

$$Sensibilidad = \frac{TP}{TP + FN} \tag{14}$$

Un buen modelo debe tener buena precisión y exhaustividad: mientras más altos ambos valores, mejor será el modelo. Desafortunadamente, ambos valores no pueden ser máximos: si se incrementa la precisión, disminuirá la exhaustividad y viceversa. Conviene entonces llegar a un punto en el que ambas medidas estén compensadas. Una métrica que combina el resultado de estas dos medidas anteriores en una es el valor F1. Así resulta mucho más fácil comparar el rendimiento de dos clasificadores.

IV. Valor F1 (F1-Score)

El valor F1 es la media armónica de la precisión y la exhaustividad. Mientras que la media normal trata todos los valores por igual, el valor F1 da más peso a los valores más bajos. Como resultado, el clasificador solo obtendrá un valor F1 alto si tanto la precisión como la exhaustividad son altos [40].

$$F1 - Score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}} \tag{15}$$

Dependiendo de la tipología de problema que se esté resolviendo, puede ser que se esté más interesado en obtener mayor grado de precisión que de exhaustividad o viceversa.

V. Exactitud (Accuracy)

Esta medida padece los mismos problemas que la precisión a la hora de evaluar el desempeño de un modelo, ya que puede mostrar un modelo mejor de lo que realmente es. La exactitud se calcula con la siguiente fórmula:

$$Accuracy = \frac{TP + TN}{TP + +TN + FP + FN} \tag{16}$$

VI. Tasa de falsa alarma (False alarm rate)

También llamada tasa de falsos positivos, es la proporción de predicciones negativas consideradas erróneamente como positivos entre el total de instancias negativas. Mientras más bajo sea este valor, mejor [67]:

$$FAR = \frac{FP}{FP + FN} \tag{17}$$

VII. Curva ROC

La curva de característica operativa del receptor se suele utilizar sobre todo en clasificadores binarios, aunque también se puede usar cuando se tienen varias clases. En este caso, se realiza una comparación por pares enfrentando una clase contra el resto (One vs. All). La curva ROC traza la exhaustividad (o TPR) contra la tasa de falsos positivos (FPR, ratio de instancias negativas que son incorrectamente clasificadas como positivas). Como con la precisión y la exhaustividad, vuelve a existir una relación inversamente proporcional entre ambas medidas: mientras mayor sea la exhaustividad, menor número de falsos positivos produce el clasificador [40].

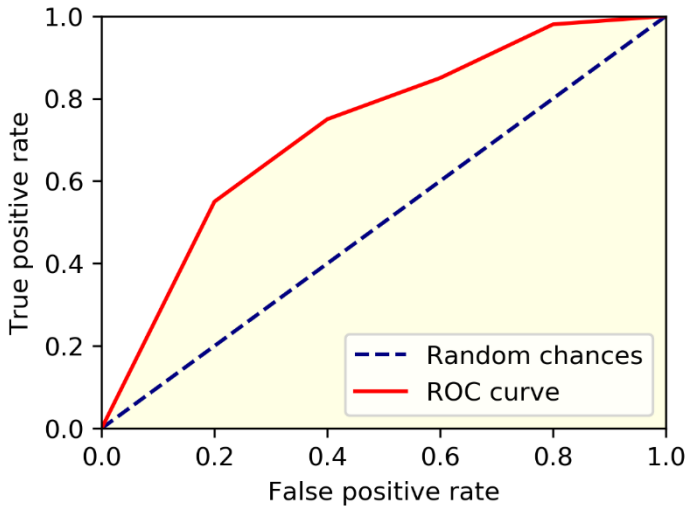


Figura 20: Ejemplo de curva ROC. Fuente: towardsdatascience.com

La línea discontinua representa la curva ROC de un clasificador totalmente aleatorio; un buen clasificador está lo más lejos posible de esta línea, moviéndose hacia la esquina superior izquierda. Se busca que el clasificador se acerque lo máximo posible a este punto, ya que es el que denota una clasificación perfecta. Una manera de comparar los clasificadores es medir el área por debajo de la curva (AUC). Un clasificador perfecto tendrá un AUC igual a 1, mientras que un clasificador puramente aleatorio tiene un AUC igual a 0.5.

3 BASES DE DATOS RELATIVAS A LA DETECCIÓN DE INTRUSOS

En la actualidad, existe una gran variedad de bases de datos públicas dedicadas a la detección de intrusos. Desde 1998 con la aparición de la KDD'99, se han sucedido otras muchas que han ido solucionando problemas encontrados en bases de datos anteriores. Por ejemplo, uno de los contratiempos más comunes de las bases de datos antiguas es que carecen de ataques que se han descubierto más recientemente, de ahí la necesidad de una actualización de su contenido. Por otro lado, también se menciona como problema la escasez de datos relativos a cierto tipo de ataques menos frecuentes.

En este apartado se plantea una revisión de una gran diversidad de bases de datos relativas al estudio de este propósito. De todos los bancos de datos presentados a continuación, existen algunos enfocados en cierto tipo de ataques u otros centrados en redes IoT o malware para Android. Se destacan 5 por encima del resto, al tratarse de bases de datos más conocidas y usadas en estudios.

3.1 KDD'99 (1998-99)

La base de datos KDD'99 ha sido la base de datos por antonomasia durante mucho tiempo para la evaluación de métodos de detección de intrusos. Este set de datos está construido a partir del set de datos del Programa de Evaluación de Detección de Intrusos llevado a cabo por la Agencia de Proyectos de Investigación Avanzados de Defensa de E.E.U.U (más conocida por sus siglas en inglés, DARPA) en 1998 [68]. El laboratorio Lincoln, dependiente del MIT, construyó un entorno que simulaba una LAN típica de la Fuerza Aérea Estadounidense, el cual procedieron a hostigar con diversos ataques.

Los datos en crudo destinados a entrenamiento resultaron ser casi 4 GB de datos TCP en crudo, equivalentes a 7 semanas de monitorización de la red. Se registraron en torno a 5 millones de conexiones, cada una de 100 bytes. Por otro lado, los datos destinados a test alcanzaron casi 2 millones de conexiones. Los ataques simulados en dicho entorno pertenecen a una de las siguientes 4 categorías: DoS, U2R, R2L y Probe. Destaca el hecho de que en los datos de test existen ataques específicos no recogidos en los datos de entrenamiento, lo cual dota de mayor veracidad y realismo a la tarea de detección de intrusos [69].

Las características de la KDD'99 pueden clasificarse en estos 3 grupos:

1. **Características básicas:** Esta categoría engloba todos los atributos que se pueden extraer de una conexión TCP/IP (duración de la conexión, servicio, protocolo, número de bytes de la fuente al destino y viceversa, entre otros).

2. **Características de tráfico:** Esta categoría incluye características que son calculadas con respecto a un intervalo temporal y se divide a su vez en dos grupos:

    a. *Características del mismo host ("same host features"):* Examina solo las conexiones en los 2 últimos segundos que tienen el mismo host de destino que la conexión actual.
    b. *Características del mismo servicio ("same service features"):* Examina solo las conexiones en los 2 últimos segundos que tienen el mismo servicio que la conexión actual.

3. **Características de contenido:** Estas características están destinadas a poder detectar ataques como los R2L y los U2R, que van incrustados en parte de los paquetes enviados, y normalmente implican una única conexión. Entonces, en pos de detectar estos comportamientos maliciosos es necesario recopilar características como, por ejemplo, el número de intentos de sesión fallidos.

Sin embargo, se han realizado diversos estudios que ponen de manifiesto la existencia de problemas inherentes a la KDD'99, como consecuencia de ser un banco de datos construido a partir de la base de datos desarrollada por DARPA. En el año 2000, el profesor de la Universidad de Carnegie Mellon John McHugh puso de manifiesto diversos errores encontrados en la construcción de la base de datos de DARPA y que, inherentemente, afectan también a la KDD'99 [70]:

- En favor de preservar la privacidad, los expertos decidieron sintetizar el tráfico normal registrado y los datos de los ataques durante la evaluación, y se afirmó que el conjunto de datos generado artificialmente era muy similar al observado en diversas bases de la Fuerza Aérea Estadounidense durante algunos meses. Sin embargo, no se llevó a cabo ninguna validación analítica ni experimental de las características de falsa alarma de los datos. No aparece ningún tipo de caracterización detallada de los datos, ni tampoco se justifica que los sistemas que usen esta base de datos como prueba deban exhibir comportamientos de falsa alarma al someterlos al tráfico normal, ya que existe tráfico en Internet que puede resultar sospechoso (datos que difieren de los originales al retransmitir, grandes cantidades de paquetes FIN y RST…) hasta que se identifica la fuente que ha originado dichos paquetes. Además, la carga de trabajo de todos los datos no parece asemejarse al tráfico existente en las redes reales.

- Los recolectores de tráfico como TCPdump, el usado para recopilar el tráfico por parte de DARPA, tienden a sobrecargarse y descartar paquetes cuando el flujo de tráfico es bastante intenso. Aun así, no hubo ningún tipo de chequeo para verificar si realmente TCPdump había descartado paquetes.

- En último lugar, tampoco existen definiciones pertinentes de los ataques. Un ataque de probing (sondeo) no tiene por qué ser un ataque necesariamente a menos que se exceda un umbral específico. En la base de datos de DARPA no se especifica ningún tipo de criterio para discernir entre si estos comportamientos pertenecen a un ataque o no.

Por último, en [69] también se especifican problemas más concretos de la KDD'99, que se suman a los enumerados en los puntos anteriores:

- Una de las deficiencias más importantes es la existencia de registros redundantes, lo que

provoca que los algoritmos de aprendizaje establezcan un sesgo hacia los registros más frecuentes en el compendio de datos. Esto hace que los ataques que ocurren con menor frecuencia no sean aprendidos por el algoritmo.

- Se realizó un experimento con distintos algoritmos de aprendizaje para que éstos etiquetasen los registros y después comparar dichos registros etiquetados por los algoritmos con el registro original. La gran mayoría de entradas de la base de datos (en torno al 98% en el set de entrenamiento y 86% en el set de testeo) fueron etiquetadas correctamente por todos los clasificadores empleados. Esto escenifica que la base de datos no pone demasiado a prueba las capacidades de clasificación de los algoritmos (no tiene demasiada dificultad para ellos).

3.2 NSL-KDD (2009)

Además de recalcar los notorios fallos de la KDD'99, Tavallaee, Bagheri, Lu y Ghorbani también presentaron la solución a muchos de los problemas encontrados, naciendo así la base de datos NSL-KDD [69]. Aunque sigue arrastrando deficiencias del banco de datos de DARPA y puede que no sea una representación perfecta de las redes reales por la falta de bases de datos que existían hace 10 años relativas al tema, es bastante válida para permitir la evaluación de distintos métodos de detección de intrusos y su posterior comparación. La Tabla 1 muestra los archivos que presenta este banco de datos.

Las mejoras que presenta la NSL-KDD son las que siguen [5]:

- Se eliminan los registros redundantes de la colección de datos, así que los clasificadores no se verán sesgados por los registros más frecuentes en el set. Tampoco existen registros duplicados en los conjuntos de datos propuestos.

- Atendiendo al experimento mencionado en el punto anterior para que varios algoritmos de aprendizaje etiquetasen los registros, se escogió un número de entradas de cada nivel de dificultad inversamente proporcional al porcentaje de registros existentes en el conjunto original de KDD'99. Por ejemplo, el número de entradas etiquetadas correctamente por 5 o por menos algoritmos de aprendizaje es el 0.04% de los registros de la base de datos, por lo que en el nuevo conjunto de datos se incluyen el 99.96% de registros de este grupo. De esta forma, las tasas de clasificación de los distintos métodos de aprendizaje varían en un rango más amplio, lo cual propicia una mejor evaluación de cada una de las técnicas.

- El número de registros que contiene este banco de datos es razonable, lo cual se traduce en una ventaja a la hora de realizar los experimentos puesto que no es necesario extraer un subconjunto de los datos para trabajar.

Archivo	Descripción
KDDTrain+.arff	Set completo de datos para el entrenamiento con etiquetas binarias y en formato ARFF.
KDDTrain+.txt	Set de datos completo para el entrenamiento, incluyendo tipos de ataques y el nivel de dificultad, en formato CSV.

KDDTrain+_20Percent.arff	Subconjunto del 20% del archivo KDDTrain+.arff.
KDDTrain+_20Percent.txt	Subconjunto del 20% del archivo KDDTrain+.txt.
KDDTest+.arff	Set de datos completo para el test, incluyendo tipos de ataques y nivel de dificultad, en formato ARFF.
KDDTest+.txt	Set de datos completo para el test, incluyendo tipos de ataques y nivel de dificultad, en formato CSV.
KDDTest-21.arff	Subconjunto del archivo KDDTest+.arff el cual no contiene registros con el nivel de dificultad 21, de un total de 21 (registros etiquetados correctamente por los 21 clasificadores del experimento).
KDDTest-21.txt	Subconjunto del archivo KDDTest+.txt el cual no contiene registros con el nivel de dificultad 21, de un total de 21 (registros etiquetados correctamente por los 21 clasificadores del experimento).

Tabla 1: Archivos del dataset NSL-KDD.

3.3 UNSW-NB15 (2015)

En 2015, Nour Moustafa creó junto con el Centro Australiano de Ciberseguridad (ACCS) una nueva base de datos llamada UNSW-NB15 para la evaluación de detección de intrusos en redes. El objetivo de esta nueva colección de registros era superar los fallos que arrastraban las bases de datos clásicas (KDD'99 en mayor medida y NSL-KDD) y actualizar la diversidad de ataques representados recogida, ya que dichos bancos de datos quedaron ligeramente obsoletos con el paso del tiempo [71].

Para generar esta nueva base de datos, se usó la herramienta IXIA PerfectStorm[4], que permitía crear un híbrido entre tráfico normal y tráfico con ataques. Dicha herramienta generaba bien tráfico normal, bien tráfico malicioso, y lo dispersaba por los nodos de la red (servidores y clientes). La herramienta tcpdump instalada en uno de los routers de la red, era la encargada de capturar todo el tráfico generado. La simulación duró 16 horas, en las que se recogieron 100 GBs de flujos de tráfico. Posteriormente, los archivos pcap generados por tcpdump fueron procesados por Argus y Bro-IDS, generando así características de los paquetes de red recogidos. Para finalizar, se introdujeron los archivos de salida de ambos programas en una base de datos SQL con el objetivo de unificar las características producidas por ambas salidas.

Dichas características generadas se pueden clasificar en 5 grupos [29]:

1. **Características de flujo:** Incluye los atributos identificativos entre hosts, como por ejemplo la IP origen del paquete, IP destino del paquete, protocolo…

2. **Características básicas:** Contiene los atributos que representan conexiones de protocolos (paquetes desde el origen/destino retransmitidos o descartados, tasa de bits por segundo en origen/destino, estado de la conexión…)

[4] https://www.ixiacom.com/products/perfectstorm

3. **Características de contenido:** Engloba las características relativas a las conexiones TCP/IP (número inicial de secuencia en la fuente, número inicial de secuencia en el destino, media del tamaño de los paquetes transmitidos por el origen/destino…)

4. **Características temporales:** Recoge los atributos temporales (tiempo de llegada entre paquetes, tiempo de configuración de la conexión TCP, etc.)

5. **Características adicionales:** Esta categoría puede dividirse a su vez en dos subgrupos: características adicionales de propósito general, y características relacionadas con conexiones, las cuales se construyen a partir del flujo de 100 conexiones basadas en el orden secuencial de la característica temporal pertinente.

Finalmente, la forma que presenta este dataset consiste en 4 archivos .CSV que contienen tanto tráfico normal como tráfico malicioso. Los tres primeros .CSV contienen 700.001 registros, mientras que el último posee 440044 entradas. Se adjuntan también la tabla de verdad de la base de datos, la lista de eventos registrados de cada tipo y la lista de atributos considerados en el conjunto de datos, con su correspondiente descripción. También se adjunta una versión reducida de la base de datos, dividida en datos de entrenamiento y de test. En [29] se realiza una comparación de la UNSW-NB15 con la KDD'99, arrojando como resultado que los métodos evaluados con la KDD'99 obtenían mayor eficiencia que con la UNSW-NB15. Este hecho refleja la complejidad de este nuevo dataset, debido a los comportamientos similares del tráfico normal y malicioso registrados en ella.

3.4 Aegean Wi-Fi Intrusion Dataset (AWID) (2016)

A diferencia de las bases de datos presentadas hasta ahora, la Aegean Wi-Fi Intrusion Dataset (a partir de ahora, AWID por sus siglas) creada en 2016 por Constantinos Kolias y Georgios Kambourakis, entre otros, se centra en la detección de ataques en las redes 802.11 [72]. Las redes Wi-Fi ofrecen una gran flexibilidad y movilidad a los usuarios; no obstante, la seguridad siempre ha sido un esfuerzo a desarrollar en las WLAN. El primer mecanismo de seguridad ideado para las redes inalámbricas (WEP) se vio comprometido poco tiempo después de su salida, al resultar relativamente sencillo atacar este mecanismo y conseguir la llave para acceder a la red. Propuestas posteriores como WPA y WPA2 resultaron más robustas y confidenciales que el protocolo de seguridad WEP; aun así, estos mecanismos son vulnerables a ataques de fuerza bruta.

Los conjuntos presentados hasta ahora no son convenientes para desarrollar algoritmos de aprendizaje de detección de intrusos centrados en redes Wi-Fi debido a la distinta naturaleza de las redes cableadas y las inalámbricas. Por el contrario, según los creadores de la AWID, esta puede aprovecharse para la investigación incluso en diferentes redes inalámbricas.

Para recopilar las entradas que forman parte de este conjunto, se creó un entorno real que reproducía fielmente una infraestructura SOHO (Small Office and Home). En dicho entorno existían equipos diversos conectados a la red (smartphones, tablets, Smart TV, portátiles), un equipo encargado de lanzar los ataques (Attacker) y por último un equipo (Monitor Node) cuyo propósito era monitorizar la red para guardar todo el tráfico producido en archivos pcap.

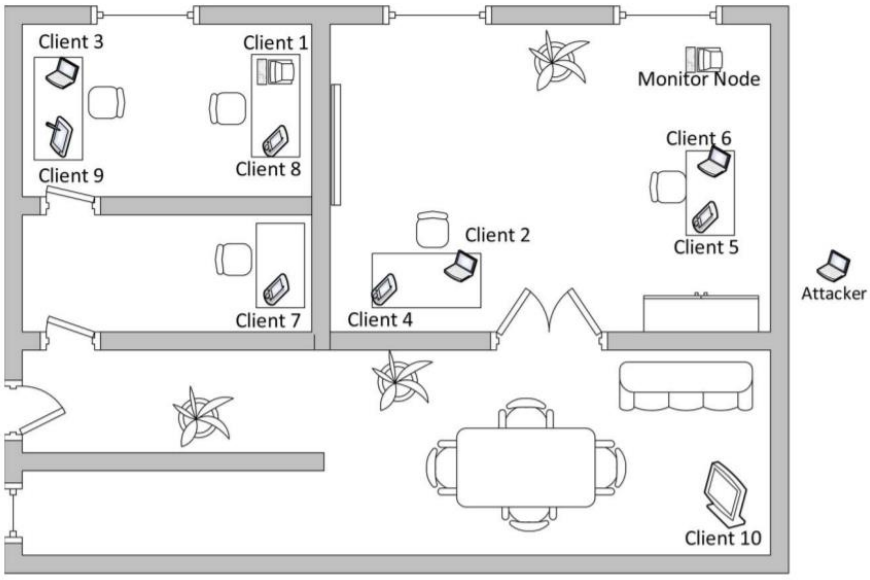


Figura 21. Mapa de posición de los elementos en el entorno SOHO.

Posteriormente, se sometían dichos archivos pcap capturados a un proceso de normalización para algunos atributos específicos (por ejemplo, convertir en enteros las direcciones MAC).

La composición de este conjunto de datos consta de dos sets que solamente difieren en el método de etiquetado: el set AWID-CLS está etiquetado en 4 clases, de acuerdo con la metodología de ejecución del acceso malicioso (ataques de inyección (injection attacks), ataques de inundación (flooding attacks), ataques de suplantación (impersonation attacks) y ataques pasivos (passive attacks)), mientras que el set AWID-ATK sigue una clasificación basada en los 16 ataques considerados. Cada uno de los sets contiene una versión completa (Full) y otra con muchas menos entradas (Reduced), los cuales a su vez contienen una versión para entrenamiento y otra para testeo. Dichas versiones reducidas no están extraídas del set completo, si no que se capturaron en otras sesiones [72].

Las versiones reducidas de entrenamiento de ambos conjuntos de datos (AWID-ATK-R-Trn y AWID-CLS-R-Trn) contienen 1.795.575 entradas, de las que 1.633.190 son tráfico normal y 162.385 son tráfico malicioso. Estos grupos de datos fueron recabados tras monitorizar durante 1 hora el tráfico del escenario. Por otra parte, las versiones completas de los conjuntos de datos contienen 37.817.835 grabaciones, de las cuales 1.085.372 son ataques a la red.

3.5 CICIDS2017 (2017)

La base de datos CICIDS fue creada en 2017 por investigadores del Instituto Canadiense de Ciberseguridad en la Universidad de New Brunswick, como consecuencia de la notoria dificultad existente entre los investigadores para encontrar datasets adecuados con los que evaluar sus técnicas de aprendizaje máquina [73]. Muchos bancos de datos destinados a este fin no se comparten a causa de problemas de privacidad. Sumado a esto, aquellas que finalmente sí se difunden son fuertemente anonimizadas, además de sufrir de problemas de diversidad de ataques y acarrear la necesidad de estar en constante revisión para incluir nuevo malware.

En 2016, Gharib y otros investigadores identificaron 11 criterios de obligado cumplimiento para crear una base de datos de detección de intrusos confiable: configuración completa de la red, banco de datos correctamente etiquetado, tráfico completo, interacción completa, captura completa, diversidad de protocolos y de ataques, anonimato, heterogeneidad, poseer un set de características exhaustivo y recopilar metadatos [74]. Hasta la aparición de la CICIDS, ninguna base de datos de las existentes cumplía estos 11 requisitos críticos, como se observa en la siguiente tabla comparativa.

	Network	Traffic	Label.	Interact.	Captu.	Protocols					Attack Diversity							~~Ano.~~	Heter.	Features	Meta.
						http	https	SSH	FTP	Email	Browser	Bforce	DoS	Scan	Bdoor	DNS	Other				
DARPA	YES	NO	YES	YES	YES	YES	NO	YES	YES	YES	NO	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES
KDD'99	YES	NO	YES	YES	YES	YES	NO	YES	YES	YES	NO	YES	YES	YES	NO	NO	YES	NO	NO	YES	YES
DEFCON	NO	NO	NO	YES	YES	YES	NO	YES	NO	NO	NO	NO	NO	YES	YES	NO	YES	-	NO	NO	NO
CAIDAs	YES	YES	NO	NO	NO	-	-	-	-	-	NO	NO	YES	YES	NO	YES	YES	YES	NO	NO	YES
LBNL	YES	YES	NO	NO	NO	YES	NO	YES	NO	NO	-	-	-	YES	-	-	-	YES	NO	NO	NO
CDX	NO	NO	NO	YES	YES	YES	NO	YES	YES	YES	NO	NO	YES	YES	NO	YES	-	-	NO	NO	NO
KYOTO	YES	NO	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	NO	NO	YES	YES
TWENTE	YES	YES	YES	YES	YES	YES	NO	YES	YES	NO	NO	YES	NO	YES	NO	NO	YES	-	-	NO	YES
UMASS	YES	NO	YES	NO	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	YES	-	-	NO	NO
ISCX2012	YES	NO	YES	YES	YES	YES	NO	YES	YES	YES	YES	YES	YES	YES	YES	NO	YES	NO	YES	NO	YES
ADFA2013	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES	YES	YES	NO	NO	YES	NO	YES	NO	-	NO	YES
CICIDS2017	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES

Figura 22. Comparación entre la CICIDS2017 y otras bases de datos atendiendo a las 11 características críticas para realizar una BBDD de detección de intrusos.

Para la construcción de este banco de datos, se construyó un entorno dividido en dos redes: la red víctima y la red atacante. Se procuró cubrir las debilidades existentes en otros escenarios construidos para crear bases de datos destinadas a este fin dotando a los dispositivos de las redes de distintos sistemas operativos, y de la infraestructura común en una red. Uno de los puertos en la red víctima fue configurado como puerto espejo para recopilar el tráfico de la red [73].

El período de captura del tráfico duró 5 días, de los que solamente uno contiene tráfico normal, mientras que los 4 restantes mezclan tráfico normal con distintos tipos de ataques a la red, en distintas horas. Para generar el tráfico se usó el sistema *B-profile* (Benign Profile System), encargado de emular el comportamiento humano a la hora de generar tráfico en la red. La herramienta *CICFlowMeter*[5] permitió la extracción de 80 características, que después fueron cribadas para elegir los atributos más adecuados a la hora de detectar cada uno de los ataques. Finalmente, todo el tráfico se agrupó en 8 archivos .CSV.

A pesar de los esfuerzos realizados por crear una base de datos completamente fiable para realizar la evaluación de distintos algoritmos, contiene ciertos defectos, como bien destacaron en un estudio de 2018 Panigrahi y Borah [75]. Primeramente, el hecho de tener el set de datos fragmentados en 8 archivos puede resultar engorroso a la hora de procesarlos. También recalcaban el hecho de que la base de datos resultaba bastante extensa, y procesar tal cantidad de datos podía ser problemático si no se posee la suficiente potencia computacional.

Asimismo, algunas entradas del dataset no están etiquetadas, o bien carecen de información en ciertos campos, por lo que habría que eliminarlas en el preprocesado de datos. Por último, se subraya el hecho de que existe un desequilibrio entre las entradas de cada una de las clases de la base de datos. Como consecuencia el clasificador se inclina hacia la clase con más entradas, mostrando menor precisión con mayor falsa alarma. Concretamente, la prevalencia de muestras de la clase mayoritaria (tráfico normal) es del 83.34%, mientras que de la clase menos frecuente (Heartbleed) es del 0.00039%.

[5] http://netflowmeter.ca/

3.6 Otras bases de datos existentes

Además de los bancos de datos mencionados hasta este momento, existen algunos otros dedicados a redes más específicas, o a un tipo concreto de ataques. En 2019, el profesor de la Universidad de Coburgo Markus Ring junto con otros colaboradores, realizó un completo estudio sobre las bases de datos existentes para detección de intrusos, haciendo acopio de un gran número de ellas y resaltando sus peculiaridades [76]. De entre todas ellas, se enumeran las siguientes:

- **LBNL (2004-2005):** Este conjunto de datos es bastante común en la investigación sobre detección de intrusos. La motivación principal de la creación de este dataset era analizar las características del tráfico de red dentro de las redes empresariales, más que por crear una nueva base de datos. Este conjunto de registros posee más de 100 horas de trafico de red en formato de paquetes, y no está etiquetado por razones de privacidad.

- **Kyoto (2006 a 2009):** Este dataset es un conjunto de datos de un honeypot que está a disposición de cualquier persona. En este caso, se utilizó el BRO IDS[6] para convertir el tráfico basado en paquetes en un nuevo formato denominado sesiones. Cada sesión engloba 24 características, de las cuales 14 contabilizan información estadística y los 10 restantes son atributos típicos basados en el flujo de tráfico (IPs, puertos o duración de las conexiones). Los datos fueron capturados durante tres años; como consecuencia el dataset contiene la friolera de unos 93 millones de sesiones.

- **Twente (2008):** Este banco de datos fue uno de los primeros en estar basados en flujo. Abarca 6 días de datos de un honeypot que ofrecía servicios HTTP, FTP y SSH. Debido a este enfoque, casi todo el comportamiento registrado en los registros es malicioso y no existen apenas registros que reflejen un comportamiento normal de un usuario cualquiera. El conjunto está disponible públicamente, aunque sin las direcciones IP, que fueron eliminadas por motivos de privacidad.

- **CDX (2009):** Esta base de datos está concebida a partir del concepto de la llamada "guerra en red". Sangster y otros investigadores crearon esta base de datos analizando las ventajas y desventajas de una competición de guerra en red que duró cuatro días. El tráfico está registrado en forma de paquetes, y contiene tanto tráfico normal como tráfico malicioso. Adicionalmente se describen los metadatos sobre la estructura de la red y las IPs, pero individualmente los paquetes no están etiquetados. Este dataset está disponible públicamente.

- **ISOT (2010):** ISOT nació de la unión del tráfico normal de red del Laboratorio de Tráfico de la compañía Ericsson en Hungría y el Laboratorio Nacional Lawrence Berkeley con el tráfico de red malicioso generado por la división francesa del proyecto Honeynet[7]. Dicho conjunto de datos se usó sobre todo para detectar redes de bots P2P. Contiene 11 GB de datos basados en paquetes.

- **ISCX2012 (2012):** ISCX fue creada capturando el tráfico en un entorno que simulaba una red durante una semana. Para generar el tráfico, los investigadores utilizaron un enfoque dinámico para generar tráfico tanto malicioso como normal. Está disponible online y contiene varios tipos de ataques distintos.

- **CICDoS (2012-2017):** Creada por el Instituto de Ciberseguridad Canadiense, es un banco de datos disponible al público. Está dedicado exclusivamente a los ataques DDoS, de ahí que los creadores

[6] https://seguridadyredes.wordpress.com/2011/03/23/bro-ids-un-sistema-de-deteccion-de-intrusiones-basado-en-politicas-especializadas/
[7] https://www.honeynet.org/

ejecutaran 8 tipos de ataques DoS diferentes en la capa de aplicación. El tráfico normal se generó combinando el tráfico resultante del experimento comentado anteriormente con el tráfico normal del ISCX 2012. Contiene 24 horas de tráfico de red en total.

- **SSHCure (2013-2014):** Este dataset surgió a raíz del desarrollo de la herramienta SSHCure, para detectar ataques SSH. Para evaluarla, los creadores de dicho software recopilaron dos conjuntos de datos de un mes de registros dentro de una red universitaria. El tráfico no está directamente etiquetado, si no que los autores proporcionan archivos de registro con los que se puede verificar si la conexión SSH tuvo éxito o no.

- **UGR'16 (2016).** Este set de datos fue desarrollado por investigadores pertenecientes al Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada. Se centra en la captura de datos en el entorno de un proveedor de servicios (ISP). Las IP son anónimas, y los flujos de tráfico están etiquetados como normales, de fondo o de ataque. Se ejecutaron varios ataques distintos. Está disponible públicamente.

- **PUF (2018):** El PUF contiene solamente conexiones DNS recopiladas durante tres días en un campus. Todos los flujos están etiquetados usando los registros de un sistema de prevención de intrusiones; sin embargo, las direcciones IP se eliminaron de las entradas para mantener la privacidad de los usuarios. Está disponible para su uso.

4 RANDOM FOREST COMO CLASIFICADOR PARA SISTEMAS DE DETECCIÓN DE INTRUSOS

Una vez presentados los conceptos más elementales del aprendizaje máquina, el siguiente paso consiste en llevarlos a la práctica. En este capítulo se aborda el desarrollo de 2 clasificadores (ambos bosques aleatorios) destinados a detectar ciberataques pasando por los apartados descritos detalladamente en el punto *Flujo de trabajo del diseño de un algoritmo de ML* del Capítulo 2, desde la elección de la base de datos hasta la evaluación del rendimiento del modelo. Concretamente, se seguirá la hoja de ruta de dos artículos distintos que trabajan con la misma base de datos y, en última instancia, se compararán los mejores resultados obtenidos con los conseguidos por los autores. De forma general, cada uno de los puntos del flujo de trabajo de ML contará con dos subapartados, dedicados al desarrollo de los pasos descritos en cada uno de los artículos para ese punto en concreto. Los artículos utilizados son:

M. A. Umar and C. Zhanfang, *Effects of Feature Selection and Normalization on Network Intrusion Detection*. 2020 [67].

N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of Machine Learning Algorithms for Anomaly Detection," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pp. 1–8, doi: 10.1109/CyberSecurity49315.2020.9138871 [77].

Por cuestiones de comodidad, a lo largo del capítulo se aludirá a ellos como Artículo I y Artículo II, respectivamente. En ambos artículos se trabaja con la base de datos UNSW-NB15, que será tomada como referencia para testar todos los modelos realizados en este trabajo.

4.1 Recopilación de datos y visualización de la base de datos

El primer paso para llevar a cabo un modelo de *machine learning* es hacerse con datos suficientes tanto para entrenar el modelo como para testarlo. En este caso, se evaluará el modelo de aprendizaje máquina con la base de datos UNSW-NB15, por tratarse de una de las más recientes y, a su vez, de las más completas que existen por el momento. Como primera fase, se procede a tratar con mayor detalle la estructura de información de este dataset, para esbozar unas primeras conclusiones del conjunto de datos con el que se va a trabajar posteriormente. El código que realiza todo este trabajo es *visualizacion_sets.py*[8]. El equipo con el que se ejecutan los experimentos se compone de un procesador Intel i7-9750HF a 2.60 GHz y hasta 4.5 GHz con Turbo, 6 núcleos, 16 GB de RAM y GPU Nvidia GeForce GTX 1650.

[8] *Véase Anexo B.*

4.1.1 UNSW-NB15

Como se presentó en el Capítulo 3, esta base de datos consta de 4 archivos .CSV donde se encuentra todo el tráfico registrado, ascendiendo el número de instancias del set de datos completo a 2.540.047 registros. A su vez, también dispone de un conjunto más reducido de datos (UNSW_NB15_training-set.CSV y UNSW_NB15_testing-set.CSV) para evaluar los modelos de aprendizaje. El primer archivo contiene 175.342 registros, mientras que el segundo posee 82.332 entradas. Para mayor facilidad a la hora de visualizar las estadísticas de este set reducido, se agrupan ambos .CSV en uno solo, denominado UNSW-NB15_red.CSV. Se hace también lo propio con los 4 .CSV del conjunto completo de datos, fusionándose todos ellos en el archivo UNSW-NB15_full_label.CSV. Las características que presenta la base de datos UNSW-NB15 aparecen registradas en la tabla que se muestra a continuación [78]:

Nº	Nombre	Tipo	Descripción
1	id**	Entero	Identifica numéricamente la entrada (1,2,3…)
2	srcip*	String	Dirección IP del origen
3	sport*	Entero	Puerto origen
4	dstip*	String	Dirección IP del destino
5	dsport*	Entero	Puerto destino
6	proto	String	Protocolo usado
7	state	String	Estado del protocolo
8	dur	Float	Duración del registro
9	sbytes	Entero	Bytes de origen a destino
10	dbytes	Entero	Bytes de destino a origen
11	sttl	Entero	Valor TTL origen-destino
12	dttl	Entero	Valor TTL destino-origen
13	sloss	Entero	Paquetes origen retransmitidos o tirados
14	dloss	Entero	Paquetes destino retransmitidos o tirados
15	service	String	http, ftp, smtp, ssh, dns, ftp-data, irc y – si el servicio no es muy común
16	sload	Float	Bits/s origen
17	dload	Float	Bits/s destino
18	spkts	Entero	Número de paquetes origen-destino

19	dpkts	Entero	Número de paquetes destino-origen
20	swin	Entero	Tamaño de ventana TCP origen
21	dwin	Entero	Tamaño de ventana TCP destino
22	stcpb	Entero	Nº secuencia inicial TCP origen
23	dtcpb	Entero	Nº secuencia inicial TCP destino
24	smean	Entero	Media del tamaño de los paquetes tx por el origen
25	dmean	Entero	Media del tamaño de los paquetes tx por el destino
26	trans_depth	Entero	Conexión de la transacción petición/respuesta de http
27	res_bdy_len	Entero	Tamaño del contenido tx desde http
28	sjit	Float	Jitter del origen (en ms)
29	djit	Float	Jitter del destino (en ms)
30	stime*	Timestamp	Comienzo de la transmisión
31	ltime*	Timestamp	Final de la transmisión
32	sintpkt	Float	Tiempo de llegada entre paquetes procedentes del origen
33	dintpkt	Float	Tiempo de llegada entre paquetes procedentes del destino
34	tcprtt	Float	Tiempo total de establecimiento de conexión TCP (suma de synack y ackdat)
35	synack	Float	Tiempo entre los paquetes SYN y SYN_ACK
36	ackdat	Float	Tiempo entre los paquetes SYN_ACK y ACK
37	is_sm_ips_ports	Binario	Si el paquete tiene IPs origen y destino iguales y puertos origen y destino iguales, esta variable vale 1. En caso contrario, 0
38	ct_state_ttl	Entero	Número asociado al estado (7) de acuerdo a los valores de sttl (11) y dttl (124)
39	ct_flw_http_mthd	Entero	Número de métodos como (como GET, POST…) en el servicio http
40	is_ftp_login	Binario	1 si se accede a la sesión FTP con usuario y contraseña; 0 en caso contrario
41	ct_ftp_cmd	Entero	Nº de flujos que incluyen un comando en la sesión FTP

42	ct_srv_src	Entero	Número de registros con mismo servicio e IPorigen en los últimos 100 registros
43	ct_srv_dst	Entero	Número de registros con mismo servicio e IPdest en los últimos 100 registros
44	ct_dst_ltm	Entero	Número de registros con mismo puerto destino en los últimos 100 registros
45	ct_src_ltm	Entero	Número de registros con mismo puerto origen en los últimos 100 registros
46	ct_src_dport_ltm	Entero	Número de registros con mismo puerto destino e IPorigen en los últimos 100 registros
47	ct_dst_sport_ltm	Entero	Número de registros con mismo puerto origen e IPdestino en los últimos 100 registros
48	ct_dst_src_ltm	Entero	Número de registros con misma IPorigen e IPdestino en los últimos 100 registros
49	attack_cat	String	Tipo de ataque: Fuzzers, Analysis, Backdoor, DoS Exploit, Generic, Reconaissance, Shellcode, Worms
50	label	Binario	0 si tráfico normal, 1 si ataque
51	rate**	Float	Tasa de envío de paquete

*Features que sólo aparecen en el dataset completo.
**Features que sólo aparecen en el set reducido.

Tabla 2: Features del tráfico del dataset UNSW-NB15. En naranja aparecen los atributos de flujo, en amarillo, los atributos básicos, en azul, los atributos de contenido; en verde, las características relativas al tiempo, en rosa, características adicionales; en verde agua se representan las caraterísticas de etiqueta.

De las 51 características de la tabla anterior, se eliminan del conjunto de datos completo las características *srcip*, *sport*, *dstip* y *dsport* por no aparecer reflejadas en ninguno de los estudios que usan esta base de datos, quedando finalmente ambos conjuntos de datos con 45 características, 43 si no se cuentan las dos columnas relativas a las etiquetas de clase *label* y *attack_cat* (la única diferencia es la presencia de *stime* y *ltime* en el set completo y *rate* e *id* en el set reducido).

Se efectúa también un recuento del tráfico normal y tráfico malicioso existente en ambos conjuntos de datos (completo y reducido). En cuanto al set completo, aprovechando la etiqueta *label* se cuentan 2.218.764 registros de tráfico normal, mientras que existen 321.283 entradas pertenecientes a tráfico anormal (un ataque por cada 7 registros de tráfico normal, aproximadamente). En el caso del set reducido, el montante de registros de tráfico normal asciende hasta las 93.000 entradas, siendo la cantidad correspondiente a los ataques algo menos del doble de instancias con respecto al tráfico normal (164.673). En las siguientes tablas se muestran con mayor grado de detalle los números pertenecientes a cada uno de los conjuntos de datos:

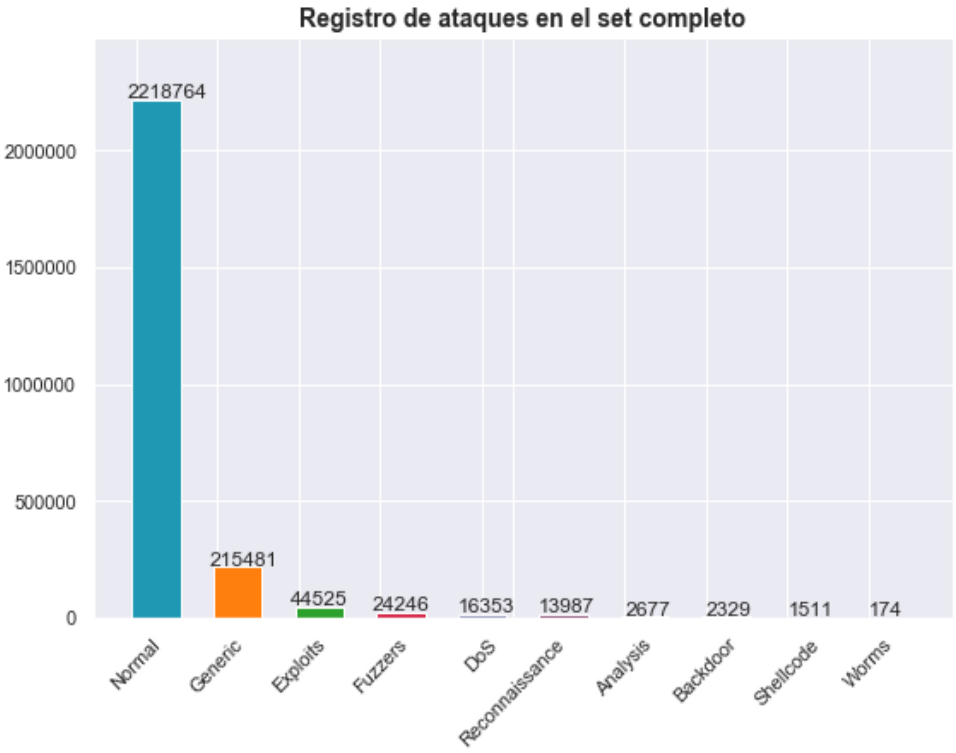


Figura 23: Recuento de instancias por categoría en el set de datos completo.

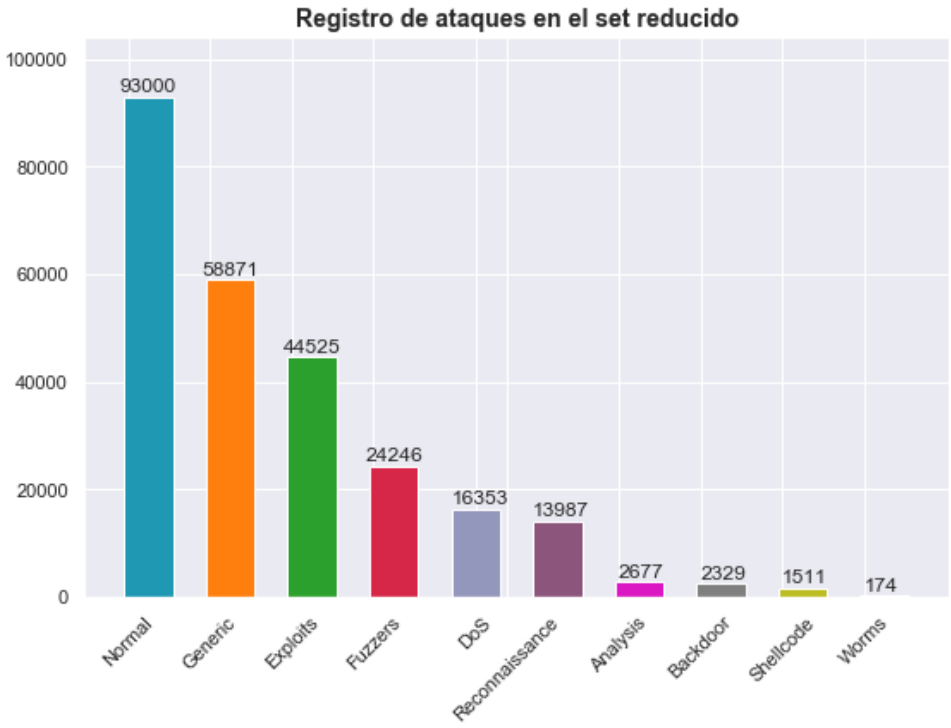


Figura 24: Recuento de instancias por categoría en el set de datos reducido.

Queda patente el hecho de que el conjunto de datos completo es imbalanceado, ya que el porcentaje de ataques frente al tráfico normal es muy pequeño en algunas clases. Este fenómeno se acusa aún más en el conjunto completo de registros, donde el número de entradas de la clase mayoritaria (tráfico normal) llega a ser incluso 10000 veces mayor que la cantidad existente de muestras de otras clases (*worms*). No obstante, en una situación real esto sería un ejemplo del tráfico que nos podemos encontrar: la gran mayoría de flujos de paquetes monitorizados serían tráfico normal. El hecho de trabajar con un dataset así podría ocasionar que el algoritmo sesgase la clasificación de los datos, tendiendo a clasificar correctamente sólo los datos pertenecientes a la clase mayoritaria [79]. Si se quisiera solucionar este inconveniente, se podrían implementar cualquiera de las soluciones vistas en la sección "Balanceo de clases" del Apartado 1.4.2 del Capítulo 2.

También resulta interesante visualizar la matriz de correlación entre las distintas características del set UNSW-NB15. Las siguientes figuras muestran la matriz de correlación en el set completo y en el reducido:

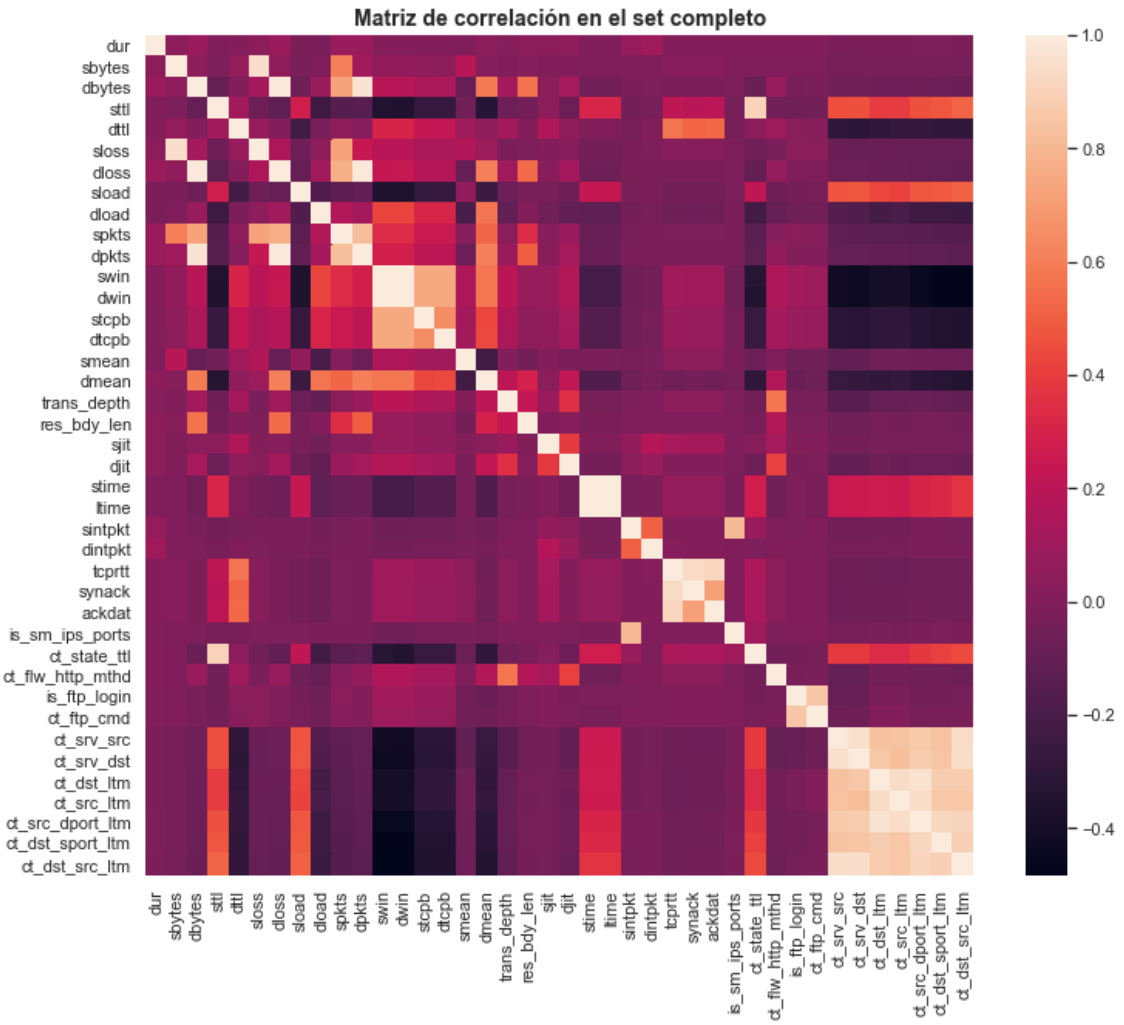


Figura 25: Matriz de correlación para el set de datos completo.

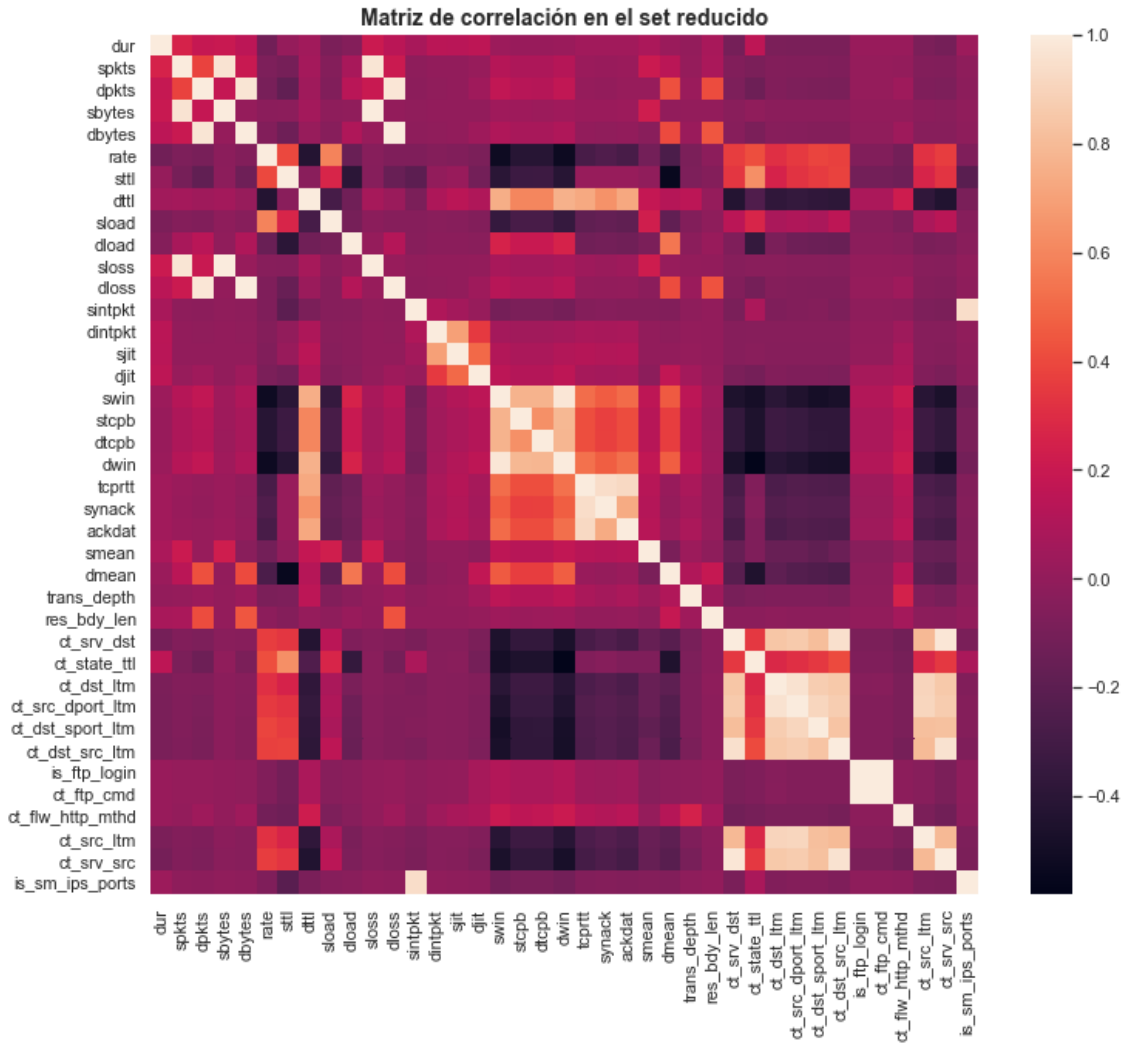


Figura 26: Matriz de correlación para el set de datos reducido.

El coeficiente de correlación se mide para establecer la relación existente entre dos características. Este valor oscila entre 1 (máxima correlación positiva; a valores altos de la primera característica le corresponden valores altos de la segunda característica) y -1 (máxima correlación negativa; a valores altos de la primera característica le corresponden valores bajos de la segunda característica). Si el coeficiente de correlación vale 0 significa que no hay ningún tipo de relación. Se observa de ambas figuras que hay varios pares de características que están bastante correlacionadas entre sí: sirvan como ejemplo los pares {sloss, sbytes}, {dloss, dbytes} o {is_ftp_login, ct_ftp_cmd}. Resulta recomendable eliminar una de las dos características de estos pares para evitar la redundancia, o bien aplicar técnicas de reducción de dimensionalidad.

Por último, se procede a visualizar un subconjunto de las instancias de la UNSW-NB15. El principal problema enfrentado a la hora de representar en una gráfica los registros de esta base de datos es la alta dimensionalidad de éstos (gran número de características). Para solventar este inconveniente, se representa el subconjunto de datos con la técnica de incrustación de vecinos estocástica distribuida o t-SNE. El resultado de aplicar este algoritmo a un subconjunto de 900 muestras tanto del set completo como del set reducido se muestra en las posteriores figuras. El número de instancias es equivalente para cada una de las categorías (100 muestras/categoría):

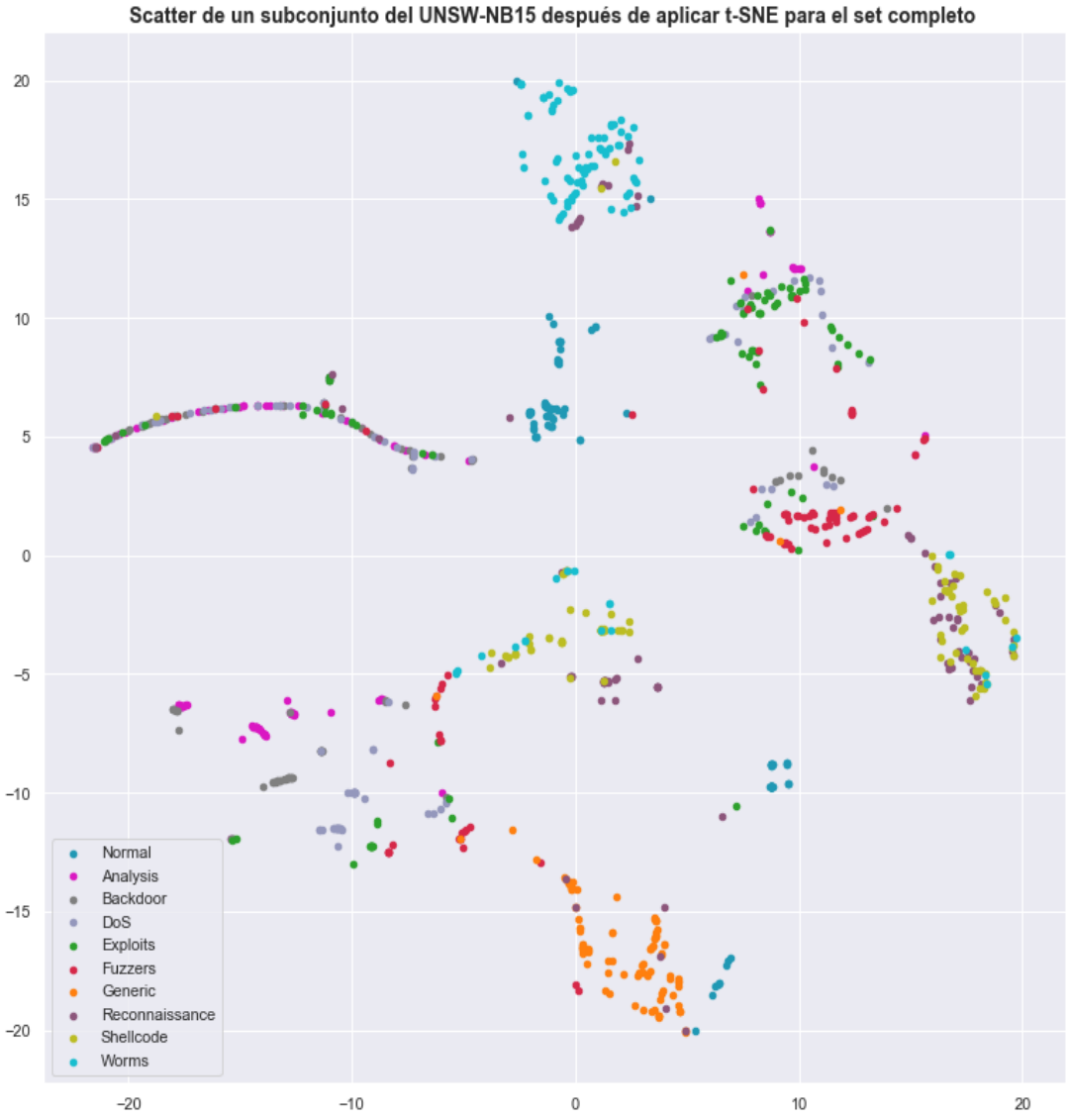


Figura 27. Resultado de aplicar t-SNE al conjunto de datos completo.

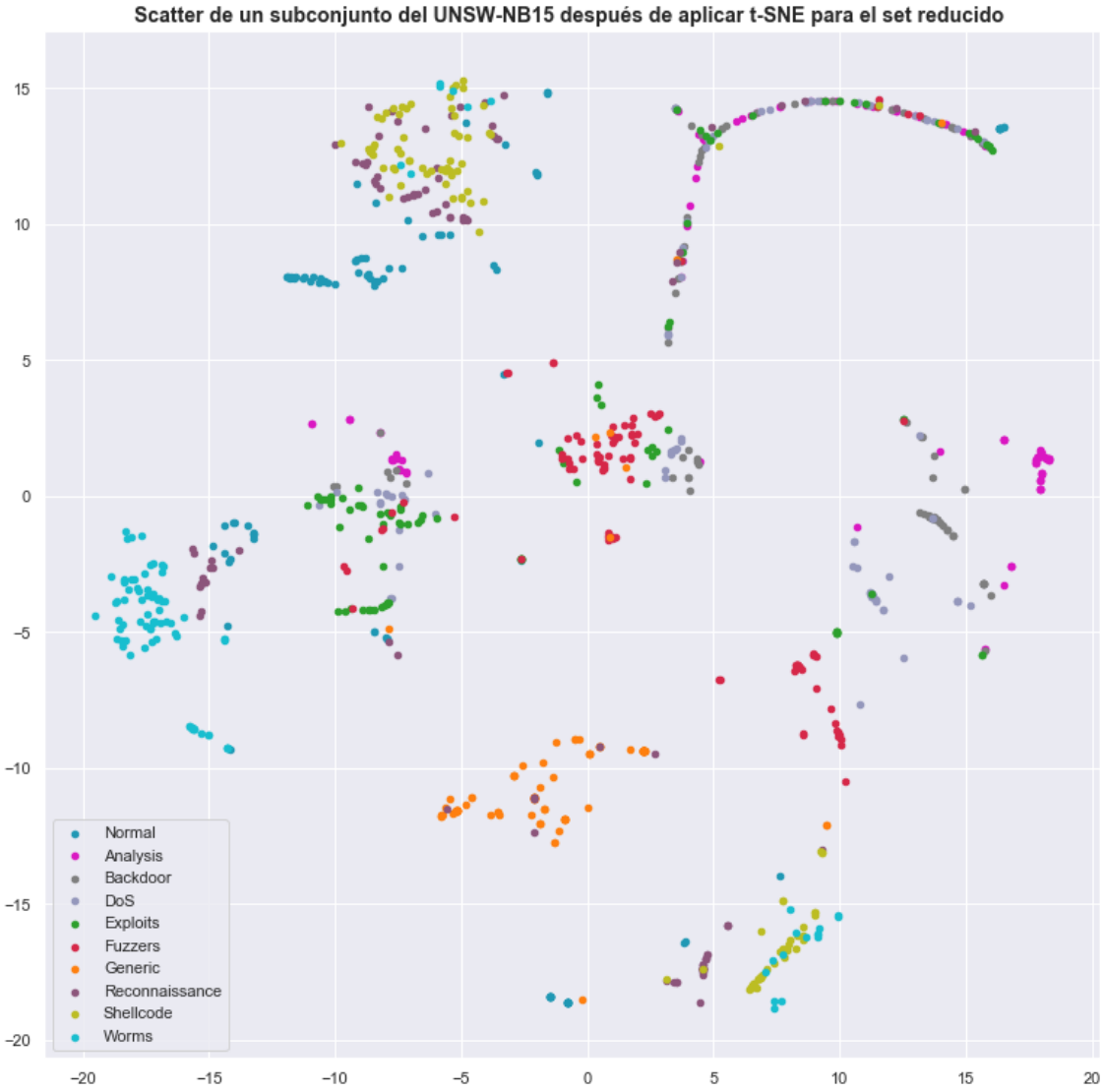


Figura 28. Resultado de aplicar t-SNE al conjunto de datos reducido.

Si se observan ambas figuras, queda patente el hecho de que existen ataques en los que casi todas sus instancias permanecen agrupadas unas cerca de otras (véase clases *normal* o *generic*). Por el contrario, hay ataques cuyos puntos se diseminan a lo largo de la gráfica solapándose con muestras de otras categorías. Esto es un dato representativo de la complejidad de este set de datos a la hora de discernir entre distintos tipos de ataques, puesto que instancias de distintas categorías se superponen unas con otras [79].

4.2 Preprocesado de los datos

En este apartado se tratarán los aspectos referentes al preprocesado aplicado a la base de datos UNSW-NB15 en los dos artículos que se han escogido como objeto de estudio. Se comentará el tratamiento realizado a los datos (normalización, codificación, limpieza de valores nulos, selección de características) antes de pasar los datos al modelo de aprendizaje máquina en cada caso.

4.2.1 Artículo I [67]

El conjunto de datos con el que se trabaja en este artículo es el reducido; concretamente, sólo se utiliza el archivo UNSW_NB15_training-set.CSV. Una parte del preprocesado del conjunto de datos está realizado con el software WEKA, herramienta especialmente dedicada al análisis de datos y modelado predictivo[9].

En primer lugar, en [67] se realiza una filtración del conjunto de datos eliminando las columnas *id* y *attack_cat.* La clasificación presentada en el documento es binaria (ataque-tráfico normal), sin distinguir entre distintos tipos de ataques. Por ello, solamente se conserva la característica *label* (0 - tráfico normal, 1 - ataque). Posteriormente a borrar estos dos atributos, se procede a dividir el dataset en la proporción de datos destinada al entrenamiento (67%) y al test (33%). Para esta labor se emplea la función *Resample* de WEKA.

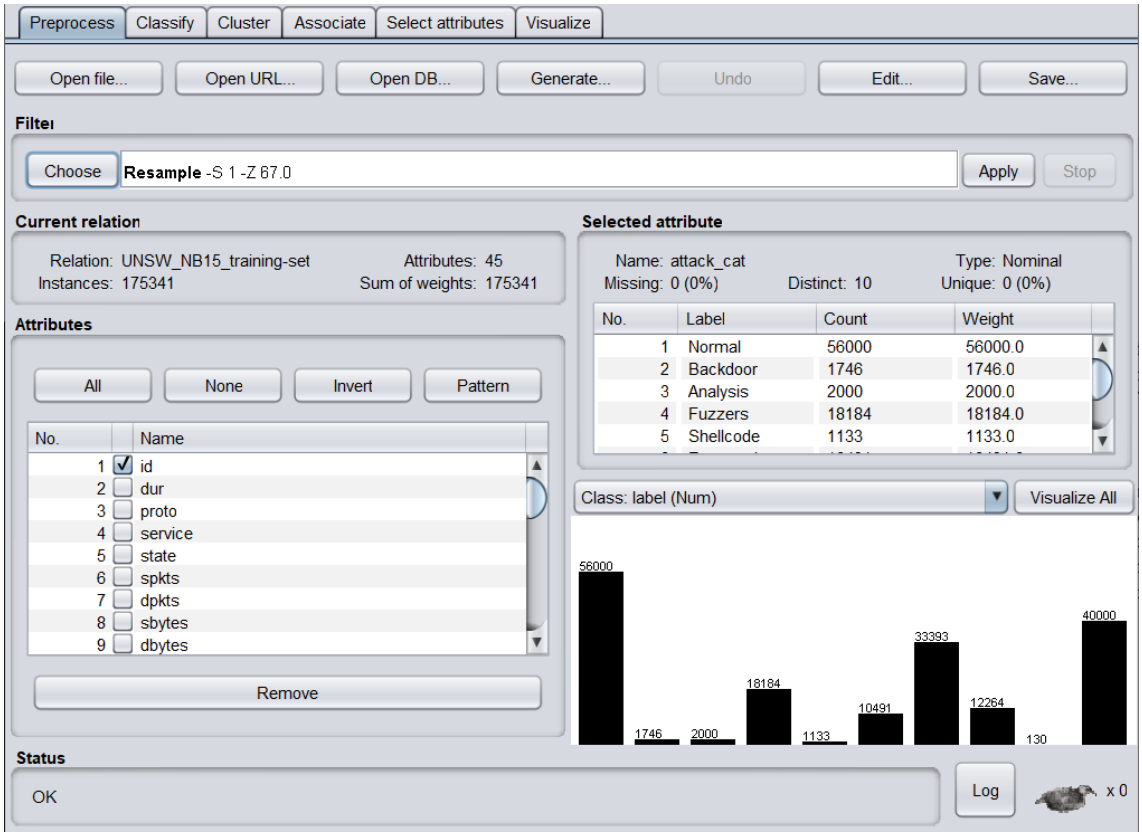


Figura 29. Entorno gráfico de WEKA. Después de cargar el .CSV del training set, se marcan las características id y attack_cat y se eliminan con "Remove". En "Choose" elegimos la función Resample para dividir el dataset en entrenamiento y test.

Para seleccionar la función *Resample,* pulsamos "Choose" > filters > unsupervised > instance > *Resample*. El atributo *sampleSizePercent* es el que determina el porcentaje del total de datos que se extraerán para la división, por eso toma el valor de 67 (%). Una vez se ha obtenido el subset de entrenamiento, se guarda en formato .ARFF. Para obtener el subset de test, hay que pulsar en *Undo* para volver a tener el conjunto completo de datos, y se marca en las opciones de *Resample invertSelection* y *noReplacement.* Una vez se tiene el conjunto de test tras pulsar en *Apply*, se guarda en formato .ARFF.

[7] *Véase Anexo A.*

Una vez obtenidos los dos subsets, el siguiente paso consiste en realizar selección de características. Se emplea un método *"wrapper"* para realizarlo, siendo un método de árbol de decisión el evaluador de características, concretamente el J48. Para llevar a cabo este paso, es necesario dirigirse a la pestaña "Select Attributes" del menú superior, posteriormente a haber cargado en la pestaña "Preprocess" > "Open File" el set de entrenamiento. En "Select Attributes" > Attribute Evaluator > WrapperSubsetEval se encuentra el algoritmo J48. Para seleccionarlo, se accede al submenú que aparece en la figura 7 clicando en la barra de "Attribute Evaluator". Los parámetros que aparecen se dejan con los valores por defecto, al igual que los parámetros del propio clasificador J48. Por último, antes de efectuar la selección de características se selecciona *label* como el atributo que contiene las dos clases que se utilizan para la clasificación.

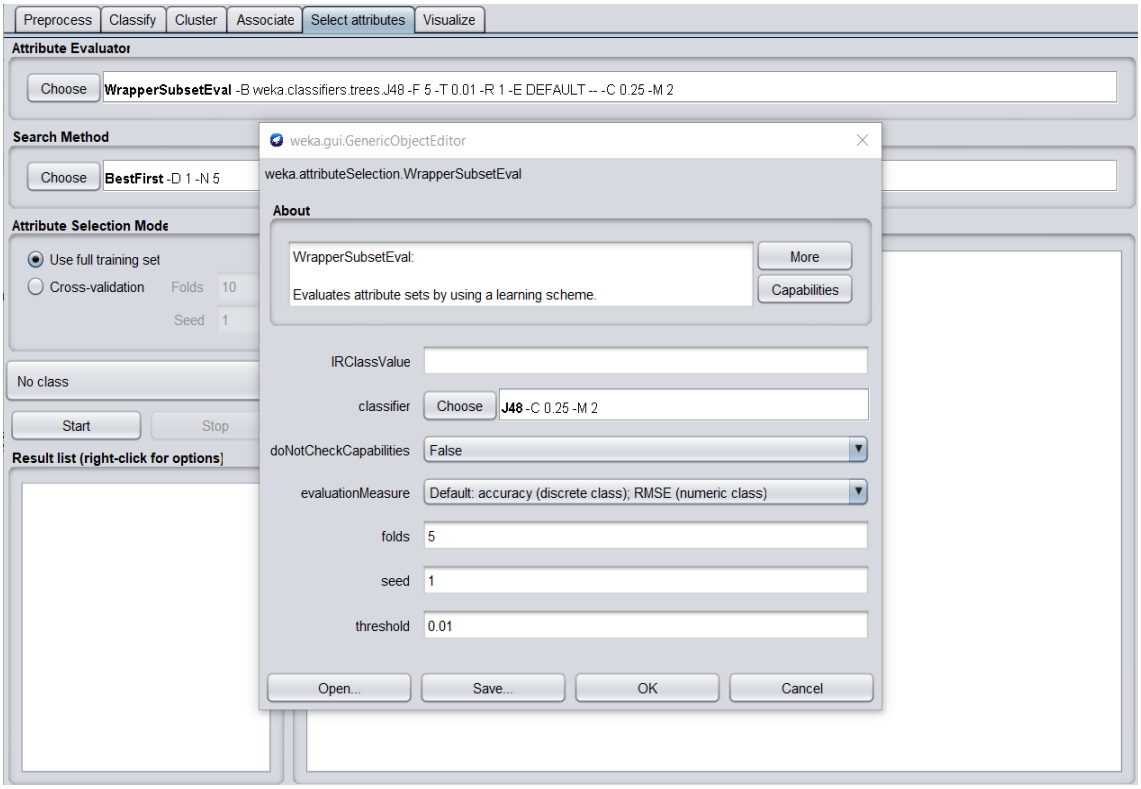


Figura 30. Submenú que aparece al clicar en la barra blanca de "Attribute Evaluator". Se muestran distintos parámetros de personalización de la evaluación. En la pestaña "Classifier" se escoge el tipo de modelo que realizará la evaluación de las características.

Llegados a este punto se intentó replicar la selección de características de la forma anteriormente explicada, aunque sin éxito: después de más de 72 horas de funcionamiento ininterrumpido se optó por pausar el procedimiento sin llegar a extraer el subconjunto de características óptimas, por lo que se escogieron directamente las 20 características óptimas obtenidas por los autores y descritas en el artículo. La principal razón reside en la enorme cantidad de tiempo que emplean en su ejecución los métodos *wrapper* debido a su complejidad computacional. El incremento de características y número de instancias del set de datos ocasiona un aumento del tiempo de ejecución, ya que se deben de entrenar un mayor número de clasificadores.

Según el artículo, el subconjunto de *features* elegidos por el J48 son 20: *proto, service, state, spkts, sbytes, dbytes, dttl, sloss, dloss, dintpkt, sjit, smean, ct_srv_src, ct_state_ttl, ct_dst_ltm, ct_src_dport_ltm, ct_dst_src_ltm, ct_flw_http_mthd, ct_src_ltm y ct_srv_dst*.

Llegados a este punto, se seleccionan las 20 *features* enumeradas en el párrafo anterior, y se eliminan las características no deseadas tanto del set de entrenamiento como del set de test como se explicó anteriormente con *Remove.* Para finalizar con el preprocesado relativo al entorno WEKA, se guardan en CSV ambos conjuntos de datos, training_bin_after_fs.CSV y testing_bin_after_fs.CSV. A partir de ahora, se manipulan ambos conjuntos de datos en Python a través del entorno de desarrollo Spyder[10].

El archivo *articulo_I.py*[11] es el encargado de efectuar el resto del preprocesado y pasos sucesivos del flujo de trabajo. Tras cargar los .CSV de entrenamiento y test, se realiza una normalización MinMax de todas las variables numéricas del dataset para evitar el sesgo del clasificador hacia atributos numéricos con gran rango de valores. En último lugar, se codifican también las variables categóricas para que el clasificador pueda trabajar de forma adecuada. La codificación escogida para las características *proto, service,* y *state* es la *one-hot*, pasando finalmente de 20 características a 172. Es necesario recalcar que, antes de llevar a cabo dicha codificación, se verifica que cada uno de los conjuntos (entrenamiento y set) tiene los mismos elementos en las columnas proto, service y state. Si existe una entrada de proto en el set de entrenamiento que no se encuentra en el set de test, se añade a esta última como una columna vacía (y viceversa). Así, una vez se aplique la codificación *one-hot*, ambos grupos de instancias tendrán el mismo número de características. La razón de este gran aumento reside en que cada uno de los valores que pueden tomar las variables categóricas pasa a ser una característica más del conjunto. Por ejemplo, la característica *state* puede tomar los valores *dns, - , smtp, ssh, http, ftp-data, ftp, pop3, ssl, dhcp, radius, irc* y *snmp*. Al aplicársele la codificación *one-hot* a la característica *state*, cada uno de estos valores enumerados pasa a ser una variable numérica, que valdrá 0 ó 1 dependiendo de si esa instancia tiene ese valor o no asignado en *state*. Si una instancia tiene el valor *http* en la característica *state,* ésta valdrá 1, mientras que el resto de características valdrán 0.

4.2.2 Artículo II [77]

El conjunto de datos con el que se trabaja en este artículo es el dataset completo (los 4 ficheros .CSV). En el archivo *visualizacion_sets.py* ya se unieron los 4 .CSV en uno, resultando el archivo UNSW-NB15_full_label.CSV, con el que se trabaja en este apartado. La primera diferencia con el artículo anterior estriba en que en este artículo se trabaja íntegramente con Python (de nuevo, el código se realiza en el entorno de desarrollo Spyder). El archivo *articulo_II.py*[12] es el encargado de efectuar el preprocesado y la posterior clasificación de las instancias. La segunda diferencia notable es que en este caso se realiza tanto clasificación binaria como multiclase; ambas opciones se consideran dentro del código.

La información referente al preprocesado aplicado a la totalidad de las instancias de la base de datos UNSW-NB15 es bastante escueta en este documento. De hecho, existe información tan relevante como el porcentaje de set de entrenamiento y test que no se proporciona, así como el tipo de normalización aplicada al conjunto de datos. En lo que respecta a este apartado de preprocesado se decide aplicar una normalización estándar a las características numéricas del dataset, después de separar el conjunto de datos completo en 70% de set de entrenamiento y 30% de test. Para asegurar una mínima cantidad de datos de cada categoría en el set de entrenamiento y de test, se activa la opción stratify con la columna correspondiente a las etiquetas de cada instancia. Se menciona también el borrado de caracteres extraños, nulos o infinitos, pero no aplica para este conjunto de instancias. La salida del comando

[10] *Véase Anexo A. Se usa la versión 3.7.6 de Python para desarrollar el código, así como la versión 1.18.1 de Numpy y la versión 1.0.1 de Pandas.*
[11] *Véase Anexo B.*
[12] *Véase Anexo B.*

dataset.isnull().values.any(), que comprueba si alguno de los valores de la base de datos es nulo da False. Además de ejecutar estos pasos en el dataset, se convierten las variables categóricas *proto, service* y *state* a columnas numéricas usando codificación *one-hot*, finalizando con 204 *features*. Es necesario recalcar que, antes de llevar a cabo dicha codificación, se verifica que cada uno de los conjuntos (entrenamiento y set) tiene los mismos elementos en las columnas *proto, service* y *state*, al igual que en el modelo anterior.

4.3 Elección del modelo, entrenamiento y evaluación

Esta sección versa sobre el modelo elegido para realizar el posterior entrenamiento y evaluación con la UNSW-NB15. En ambos artículos se prueban y contrastan resultados de muchos modelos de aprendizaje máquina, siendo el *random forest* el clasificador elegido en el presente trabajo, común a ambos documentos. Este modelo de *machine learning* es un algoritmo combinado, esto es, combinan modelos más simples para ofrecer un modelo mucho más potente.Existen muchas maneras de ensamblar algoritmos para formar otros, siendo las más usadas y populares el *boosting* (se utilizan los modelos simples de forma secuencial, haciendo que los modelos simples posteriores vayan teniendo en cuenta los errores cometidos por los modelos simples anteriores; mejorando así el rendimiento general) y el *bagging* (los métodos simples son usados en paralelo; se promedia la salida de todos los modelos simples para así reducir el error) [40]. A este último grupo pertenece el *random forest*, el cual toma como modelo simple un árbol de decisión.

Como se refería en el Capítulo 2, si el árbol de decisión era lo suficientemente complejo podía tender al sobreajuste, de ahí que una manera de evitarlo sea entrenar numerosos árboles de decisión. El procedimiento del *random forest* para clasificación es el que sigue:

- Se seleccionan **k** atributos de los **m** totales y se genera un árbol de decisión con esas **k** características.
- Se vuelve a repetir el proceso de nuevo hasta generar **n** árboles, variando siempre la cantidad de **k** características elegidas para crear cada modelo. También se puede variar la cantidad de muestras de todo el conjunto de datos que se pasan al árbol de decisión (“bootstrap sample”).
- Una vez hecho esto, se hace pasar cada muestra del set de test por cada uno de los **n** árboles de decisión para que éstos etiqueten la muestra pertinentemente. Cada árbol etiquetará con una clase distinta a la instancia; la clase más numerosa de entre las **n** salidas de todos los árboles de decisión será la clase final de la instancia.

Para este apartado, el procedimiento seguido en ambos códigos de Python es el que se muestra en la figura 25.

```
tree_clf = RandomForestClassifier(n_estimators=400, n_jobs=3,verbose=1)

tiempo_inicial = time()
tree_clf.fit(X_train, Y_train)
Y_pred = tree_clf.predict(X_test)
tiempo_final = time()
tiempo_ejecucion = tiempo_final - tiempo_inicial
print('El tiempo de ejecución fue de: %s segundos' % tiempo_ejecucion)
performance(Y_test,Y_pred,label_bin)
```

Figura 31. Parte del código común a articulo_I.py y a articulo_II.py que se encarga del entrenamiento del clasificador y posterior predicción con el set de test. La función performance extrae las métricas de interés para evaluar el rendimiento del clasificador.

En primera instancia se invoca el *random forest*, importado previamente en las primeras líneas del código y se almacena en la variable tree_clf. Debido a que en ninguno de los dos documentos se especifica información relativa a los parámetros del random forest, se dejan todos por defecto en principio [13](max_depth=None, max_features='auto', min_samples_leaf=1, max_samples_split=2), exceptuando n_estimators=400 (nº de árboles de decisión que se van a entrenar), n_jobs=3 (nº de núcleos del procesador que se emplearán para entrenar y evaluar el clasificador) y verbose=1 (para mostrar información acerca de la ejecución de los comandos del clasificador). Las variables tiempo_inicial y tiempo_final están antes y después del entrenamiento y posterior predicción realizada por el clasificador respectivamente, para poder determinar el tiempo empleado por el modelo para realizar estos procesos (tiempo_ejecucion). La línea tree_clf.fit (X_train, Y_train) es la que realiza la construcción del modelo random forest con los datos de entrenamiento, mientras que Y_pred = tree_clf.predict(X_test) se encarga de evaluar el modelo con el set de test. Estas predicciones realizadas por el modelo para el conjunto de test, junto con las etiquetas de clase verdaderas, son las que se pasan al código *metricas.py* para calcular el rendimiento del clasificador.

El código *metricas.py*[14] (a través de la función *performance*) calcula las siguientes métricas para evaluar el rendimiento final del clasificador: matriz de confusión, exactitud, precisión, sensibilidad, y F1-Score; estas tres últimas de forma general para el clasificador y también particularizadas para cada clase.

4.3.1 Artículo I

En este subapartado se muestran los resultados arrojados por la función *performance* a la hora de evaluar el buen hacer del clasificador random forest con el conjunto de datos preprocesado y tratado según se indica en el Artículo I [64]. Para una clasificación binaria y los parámetros del random forest configurados según se explica en líneas anteriores (n_estimators=400, n_jobs=3, verbose=1) se consiguen los resultados que se exponen en la figura siguiente:

Estadísticas por categoría y generales:

	precision	recall	f1-score	support
1	0.8925	0.9999	0.9432	39316
0	0.9999	0.7448	0.8537	18547
accuracy			0.9182	57863
macro avg	0.9462	0.8724	0.8984	57863
weighted avg	0.9269	0.9182	0.9145	57863

Figura 32: Resultados obtenidos de la evaluación del clasificador del Artículo I. El valor "0" se corresponde con tráfico normal, mientras que el valor "1" se corresponde con tráfico anómalo (ataques). El campo "support" muestra el número de instancias existentes de cada clase.

Como se puede observar, los resultados obtenidos por este clasificador binario son buenos en líneas generales, estando por encima del 90% de media en las cuatro métricas calculadas (promedio "weighted"). Es pertinente comentar la diferencia existente entre las filas "macro avg" y "weighted avg", ya que los resultados de las métricas son ligeramente distintos dependiendo del tipo de media

[13] *El modelo random forest tiene más hiperparámetros que los aquí especificados; en este trabajo solamente se tienen en cuenta los más relevantes (son los que se nombran).*
[14] *Véase Anexo B.*

que se tome: si se elige el promedio "macro", se computa la métrica que se esté calculando sin tener en cuenta la proporción de muestras para cada categoría en el dataset, mientras que el promedio "weighted" sí que calcula la métrica considerando la proporción de instancias existentes para cada etiqueta en el conjunto de datos.

En lo referente al tiempo de ejecución, el entrenamiento y predicción de clases se realiza de forma muy rápida (menos de 1 minuto). Particularizando para cada categoría y centrándonos en el promedio "macro", la sensibilidad y el F1-Score para la categoría de tráfico normal (0) quedan por debajo del 90%. Esto es debido al porcentaje más elevado de falsos negativos en comparación con los verdaderos positivos, como se aprecia en la matriz de confusión que se muestra a continuación:

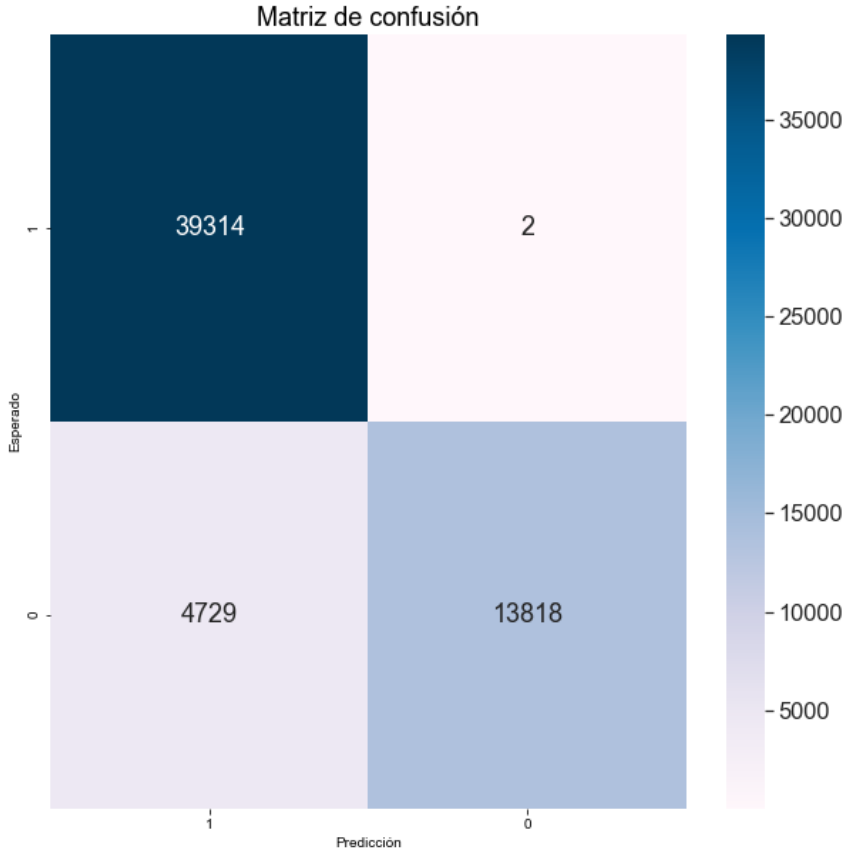


Figura 33: Matriz de confusión correspondiente a los resultados del Artículo I.

4.3.2 Artículo II

En este subapartado se muestran los resultados arrojados por la función *performance* a la hora de evaluar el desempeño del clasificador random forest con el conjunto de datos preprocesado y tratado según se indica en el Artículo II [74]. En este artículo se realiza tanto una clasificación binaria como multiclase, por lo que se mostrarán los resultados pertenecientes a ambos casos. Para una clasificación binaria y los parámetros del *random forest* configurados según se explica en líneas anteriores (n_estimators=400, n_jobs=3, verbose=1) se consiguen los resultados que se exponen en la figura siguiente:

```
El tiempo de ejecución fue: 756.940 segundos.
Estadísticas por categoría y generales:

              precision    recall  f1-score   support

           0     0.9991    0.9897    0.9944    665630
           1     0.9330    0.9939    0.9625     96385

    accuracy                         0.9902    762015
   macro avg     0.9661    0.9918    0.9784    762015
weighted avg     0.9907    0.9902    0.9903    762015
```

Figura 34: Resultados obtenidos de la evaluación del clasificador binario del Artículo II. El valor "0" se corresponde con tráfico normal, mientras que el valor "1" se corresponde con tráfico anómalo (ataques). El campo "support" muestra el número de instancias existentes de cada clase.

En este caso, los resultados obtenidos en general son mejores que en el anterior apartado, probablemente empujado por el mayor número de muestras de tráfico normal que de ataque (el hecho de que el conjunto de datos no esté balanceado se acusa más en este caso). Entrando en detalles de cada clase, el impacto de las muestras mal etiquetadas en los ataques (1) resulta mayor al tener mucho menor número de muestras que en el caso del tráfico normal; es por eso que en general se obtienen números ligeramente menores en las métricas para esta clase. Por otro lado, el tiempo de entrenamiento y predicción de clases en este caso binario ha sido de casi 13 minutos. Seguidamente se muestra la matriz de confusión de este clasificador binario:

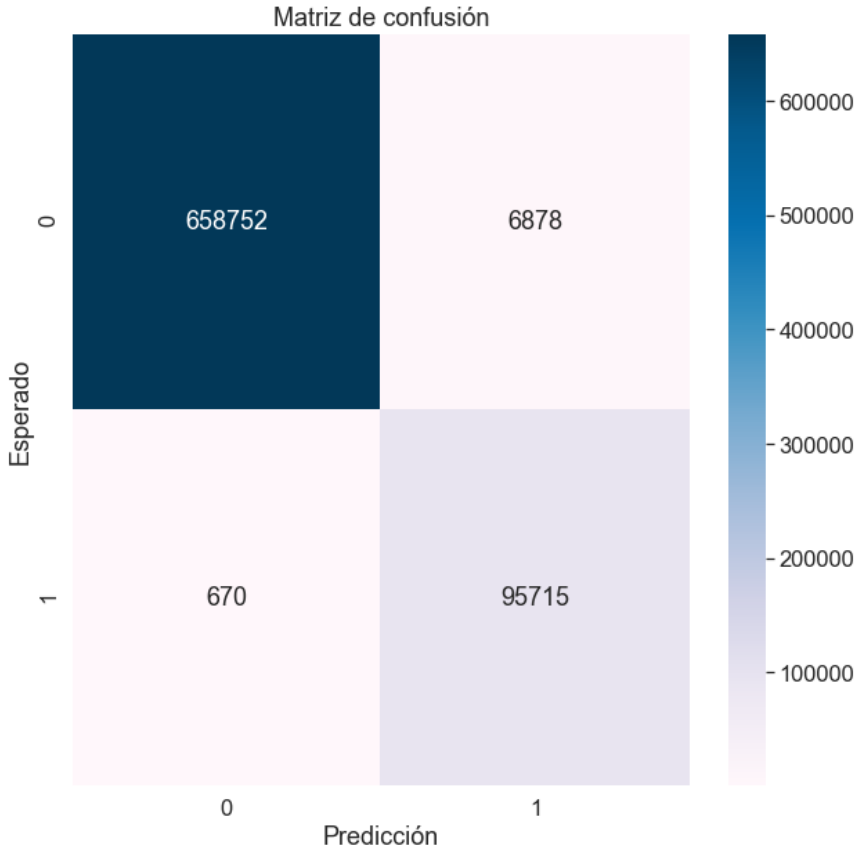


Figura 35: Matriz de confusión correspondiente a los resultados del clasificador binario del Artículo II.

Por otro lado, para una clasificación multiclase y los parámetros del random forest configurados según se explica en líneas anteriores (n_estimators=400, n_jobs=3, verbose=1) se consiguen los resultados que se exponen en la figura siguiente:

```
El tiempo de ejecución fue: 694.753 segundos.
Estadísticas por categoría y generales:

              precision    recall  f1-score   support

       Normal     0.9927    0.9974    0.9950    665630
      Generic     0.9978    0.9822    0.9900     64644
     Exploits     0.5677    0.9449    0.7093     13358
     Analysis     0.8718    0.0423    0.0808       803
Reconnaissance     0.9921    0.2402    0.3868      4196
      Fuzzers     0.6844    0.5382    0.6026      7274
          DoS     0.4759    0.0483    0.0877      4906
    Shellcode     0.5732    0.1038    0.1757       453
     Backdoor     1.0000    0.0272    0.0529       699
        Worms     0.8333    0.0962    0.1724        52

     accuracy                         0.9780    762015
    macro avg     0.7989    0.4021    0.4253    762015
 weighted avg     0.9790    0.9780    0.9743    762015
```

Figura 36: Resultados obtenidos de la evaluación del clasificador multiclase del Artículo II.

En este clasificador se aprecia claramente la dificultad que entraña la base de datos UNSW-NB15, comentada en la primera sección de este capítulo. Existen categorías para las que el clasificador funciona de forma excelente, arrojando muy buenos números (Normal, Generic) y categorías en las que no es capaz de clasificar correctamente un gran número de muestras (que coincide con las clases menos numerosas). Si se pone el foco en el promedio "weighted" las métricas son muy buenas por la misma razón que en el clasificador binario: la ausencia de balanceo de clases y el buen desempeño del clasificador en las clases más numerosas. No obstante, el promedio "macro" al tratar por igual cada una de las categorías permite visualizar un panorama más realista del rendimiento del clasificador en todas las categorías en general, consiguiendo unos resultados no demasiado elevados globalmente, como demuestra el 42% de F1-Score obtenido. La matriz de confusión se torna relevante en este clasificador multiclase para entender de forma mucho más clara los errores de clasificación cometidos por el modelo.

Como se observa en la Figura 31, existen muchos falsos positivos en categorías como Exploits o Fuzzers, así como gran cantidad de falsos negativos en clases como Reconnaissance o DoS. Los ataques menos numerosos en el conjunto de datos son los que consiguen peores resultados y los que llevan más a confusión al modelo, obteniéndose mejores métricas cuanto más numeroso es el grupo de muestras con el que cuenta la clase. Esta complejidad de distinción entre distintos tipos de ataques conlleva una disminución del rendimiento del clasificador. Además del menor número de muestras existentes de estas clases para entrenar al algoritmo se suma el hecho de que, como se observaba en la Figura 22, al aplicar la técnica t-SNE hay ataques cuyos puntos se diseminan a lo largo de la gráfica solapándose con muestras de otras categorías (exactamente, las categorías menos numerosas son las que dan este tipo de problema).

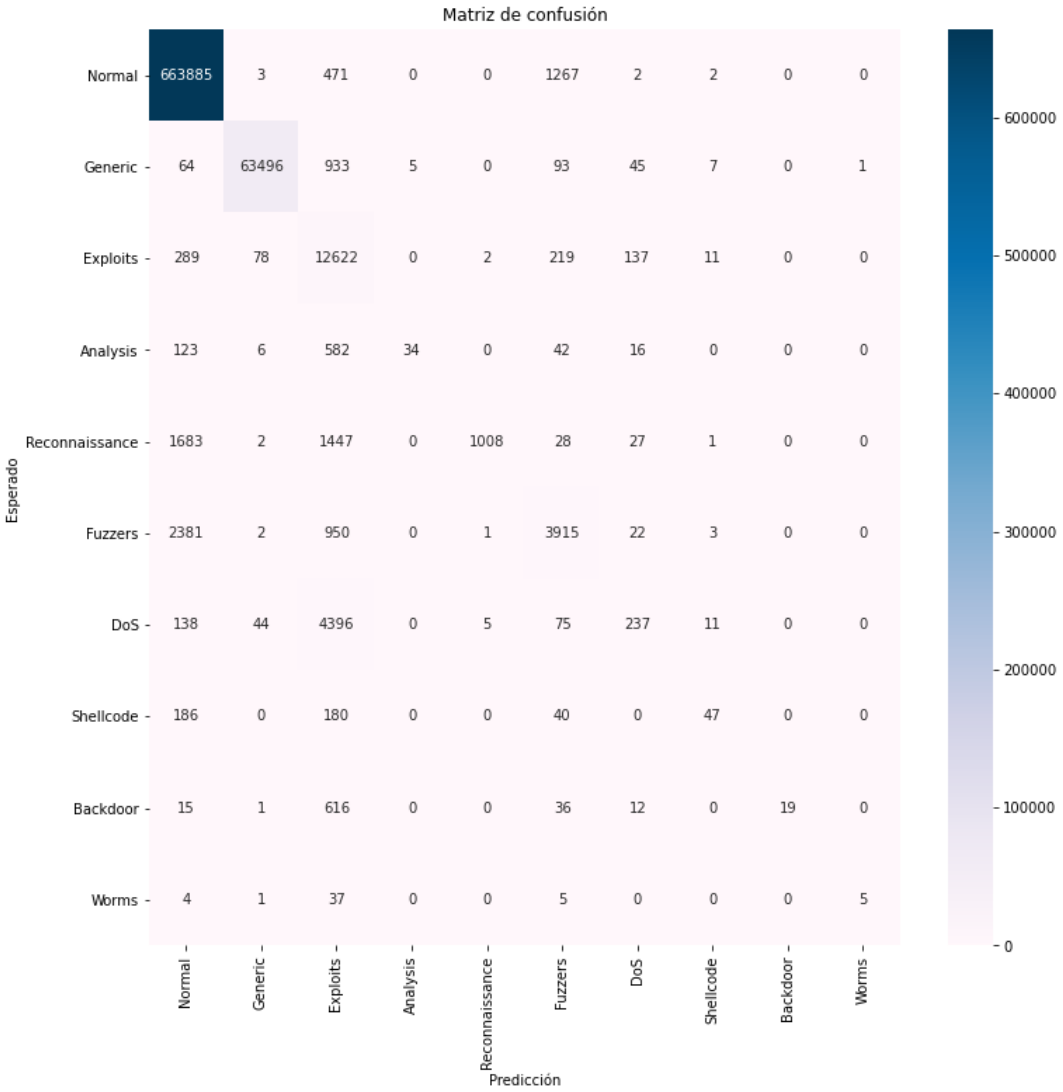


Figura 37: Matriz de confusión correspondiente a los resultados del clasificador multiclase del Artículo II.

4.4 Ajuste de hiperparámetros

Como ya se expuso anteriormente, en ninguno de los dos artículos se concretan los parámetros pasados a los clasificadores. Con objeto de intentar mejorar los resultados obtenidos en la sección anterior (sobre todo los del clasificador multiclase), se realiza una pequeña búsqueda en rejilla (*grid search*) para intentar obtener un conjunto de hiperparámetros más óptimo que el considerado inicialmente. La rejilla de parámetros considerada para la búsqueda es la que se muestra en la Figura 32.

```
param_grid = {
    'max_depth': [100, 110, None],
    'max_features': [10, 15, 'auto'],
    'min_samples_leaf': [1,40,50],
    'min_samples_split': [2,30,50],
    'n_estimators': [300, 400, 1000]
}
```

Figura 38: Rejilla para realizar el ajuste de hiperparámetros. Común a articulo_I.py y articulo_II.py

4.4.1 Artículo I

Se ejecuta la búsqueda en rejilla utilizando todos los núcleos disponibles del procesador y fijando como métrica objetivo el F1-Score. El número de validaciones cruzadas (divisiones) del conjunto de datos que hay que probar para cada combinación de hiperparámetros se fija en 4 (cv=4). El resultado de ejecutar el GridSearchCV es el siguiente:

```
Fitting 4 folds for each of 243 candidates, totalling 972 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:  6.3min
[Parallel(n_jobs=-1)]: Done 138 tasks      | elapsed: 29.5min
[Parallel(n_jobs=-1)]: Done 341 tasks      | elapsed: 70.6min
[Parallel(n_jobs=-1)]: Done 624 tasks      | elapsed: 118.3min
[Parallel(n_jobs=-1)]: Done 972 out of 972 | elapsed: 176.5min finished
La combinación de parámetros que mejor métrica objetivo arroja es:
{'max_depth': None, 'max_features': 15, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 400}
El valor del F1-Score con la mejor combinación de parámetros es:  0.9688848702848812
```

Figura 39. Resultado de la ejecución de la búsqueda en rejilla para optimizar la selección de hiperparámetros para el random forest del Artículo I.

Como se visualiza en la imagen anterior, para el mejor conjunto de hiperparámetros encontrado por la búsqueda en rejilla el F1-Score es del 96.88%. La mejor combinación de parámetros escogida resulta ser prácticamente la misma que se ha usado inicialmente, a diferencia de la configuración del parámetro max_features, que pasa de auto a 15. En comparación con los resultados obtenidos con los parámetros iniciales, se consigue una mejora de casi un 2% en el parámetro objetivo: 95.12% de F1-Score con los parámetros iniciales (macro_avg) y 96.88% de F1-Score con el mejor conjunto de hiperparámetros. Sin embargo, al llevar este conjunto de hiperparámetros al problema real, con 2/3 de datos dedicados a entrenamiento y 1/3 dedicado a test, el resultado que ofrece la métrica objetivo es prácticamente la misma que con los parámetros iniciales (en torno a un 95%).

4.4.2 Artículo II

Con el Artículo II también se intentó hacer lo propio para mejorar los parámetros usados en primera instancia, pero se desechó la opción debido a la cantidad de tiempo que empleaba la búsqueda en rejilla a causa del gran número de datos con el que se trabaja. De hecho, las prestaciones del equipo con el que trabajan los autores superan ampliamente las características del ordenador utilizado en este trabajo: 64 GB de RAM, procesador con 20 núcleos disponibles y 2 GPU's.

4.5 Mejores resultados obtenidos y comparación con resultados de los artículos

Este apartado recoge una comparación entre los resultados obtenidos con los códigos desarrollados en este trabajo y los resultados que exponen los autores en los dos artículos de referencia, comentando de forma breve las diferencias existentes entre ellos y las posibles causas.

4.5.1 Artículo I

Los autores de este documento consiguieron una exactitud del 98.51% y una sensibilidad del 99.17% aplicando selección de características al conjunto de datos y usando el algoritmo de *random forest*. Estos resultados son algo más elevados que los resultados propios, expuestos en el Apartado 1.3.1 (91.82% de exactitud y 87.24% de sensibilidad (promedio macro)).

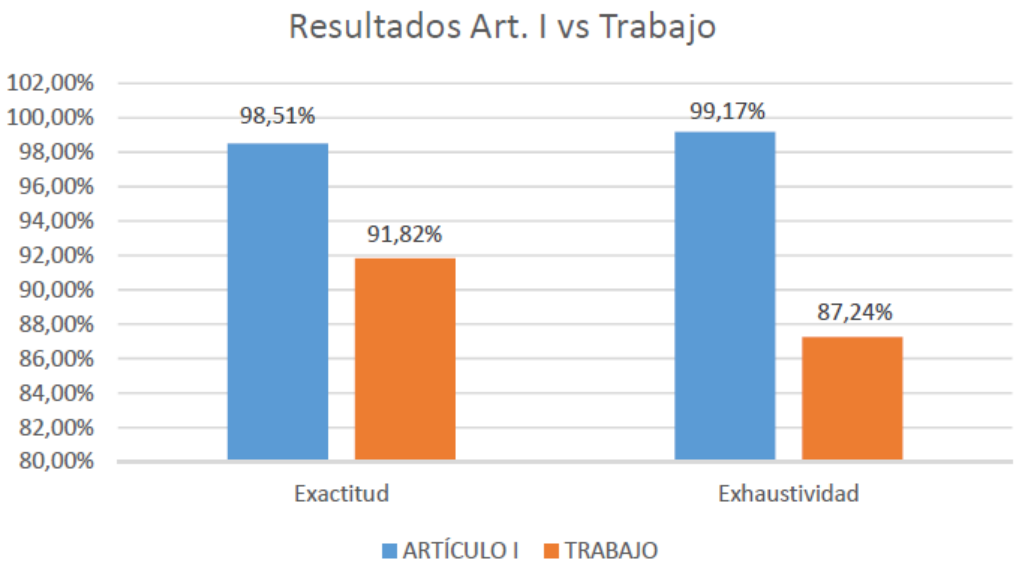


Figura 40: Comparación entre los resultados obtenidos en el presente trabajo con los expuestos en el Artículo I.

La diferencia de resultados entre el Artículo I y el trabajo quizá pueda tener su origen en que las muestras escogidas para el conjunto de entrenamiento y de test no sean las mismas. Podría ser que el conjunto de datos de test del algoritmo del Artículo I tuviese mayor número de muestras de las clases mayoritarias (las cuales el algoritmo clasifica mejor), de ahí la mejora de resultados en comparación con los obtenidos en el presente trabajo. Sin embargo, no se puede constatar del todo esta afirmación debido a que el Artículo I no ofrece información desglosada del número de muestras existentes de cada clase en el conjunto de test. En cuanto al resto, se siguen los pasos del documento uno a uno y a priori, no existe ninguna disparidad más entre el procedimiento del artículo y el desarrollado en *articulo_I.py*.

4.5.2 Artículo II

Para el clasificador binario los autores de este artículo consiguieron unos resultados globales del 87.7% de exactitud, 84.4% de precisión, 99.1% de sensibilidad y 91.2% de valor F1, mientras que los obtenidos en este trabajo son de un 99.02% de exactitud, 96.61% de precisión, 99.18% de sensibilidad y 97.84% de valor F1 (se han seleccionado los valores del promedio "macro" por guardar equidad entre todas las categorías, tengan las muestras que tengan).

Exceptuando el valor de sensibilidad, el resto de métricas están bastante por encima de los valores registrados en el documento tomado como referencia, llegando hasta a estar casi un 13% por encima aproximadamente (caso de la precisión). A priori, el desarrollo propio realizado en *articulo_II.py* sigue exactamente los mismos pasos que el documento tomado como base, aunque en este documento sí que es más notoria la falta de información como se remarcó anteriormente. Por este motivo, no se puede concluir con exactitud el motivo al cual se podría deber la diferencia existente entre ambos resultados.

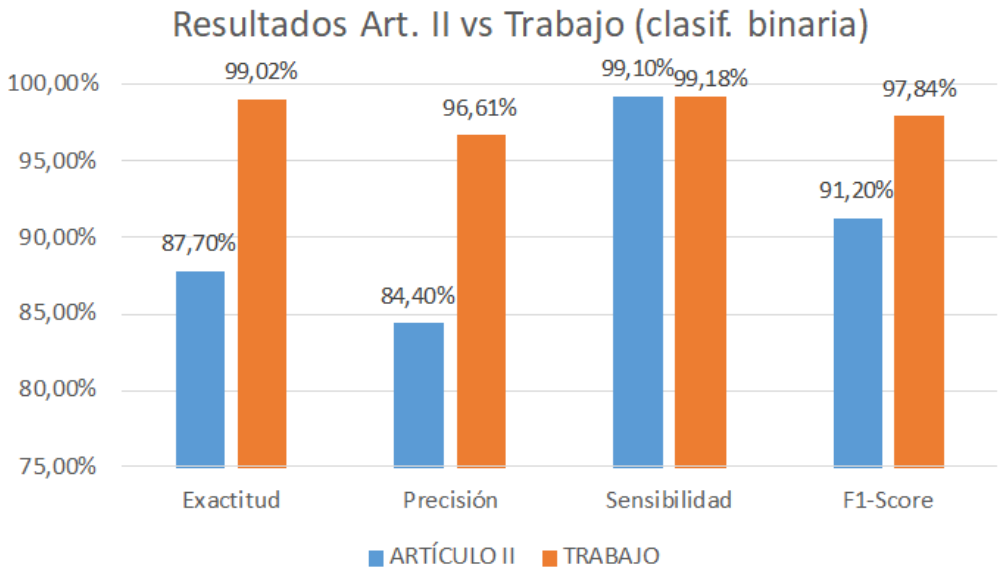


Figura 41: Comparación entre los resultados obtenidos en el presente trabajo con los expuestos en el Artículo II para el caso de clasificación binaria.

Por otro lado, para el clasificador multiclase se obtienen peores resultados globales que en el artículo de referencia. Como se visualiza en la figura 18, las métricas están en torno al 70% mientras que en el código desarrollado en este trabajo la sensibilidad y F1-Score obtenidas son de un 40.21% y 42.53% respectivamente (promedio "macro"). Hubiese resultado interesante saber de qué forma están calculadas exactamente las métricas del artículo para comparar de forma más precisa ambos resultados, ya que no se especifica en ningún momento de qué forma se calculan (promedio "macro" o "weighted"). Aun así, se toma como opción más plausible el hecho de que el global de precisión, sensibilidad y valor F1 esté calculado tomando cada una de las categorías de forma equitativa (promedio "macro", sin aplicar pesos según el tamaño de muestras existente de cada clase), ya que los resultados obtenidos en el documento original tampoco son muy altos. Tampoco se adivina una causa posible para explicar la gran diferencia existente (en este caso, a favor de los resultados del trabajo) en la métrica de exactitud: la escasez de detalles proporcionados en el Artículo II hacen que no sea posible siquiera plantear una idea que justifique dicha diferencia.

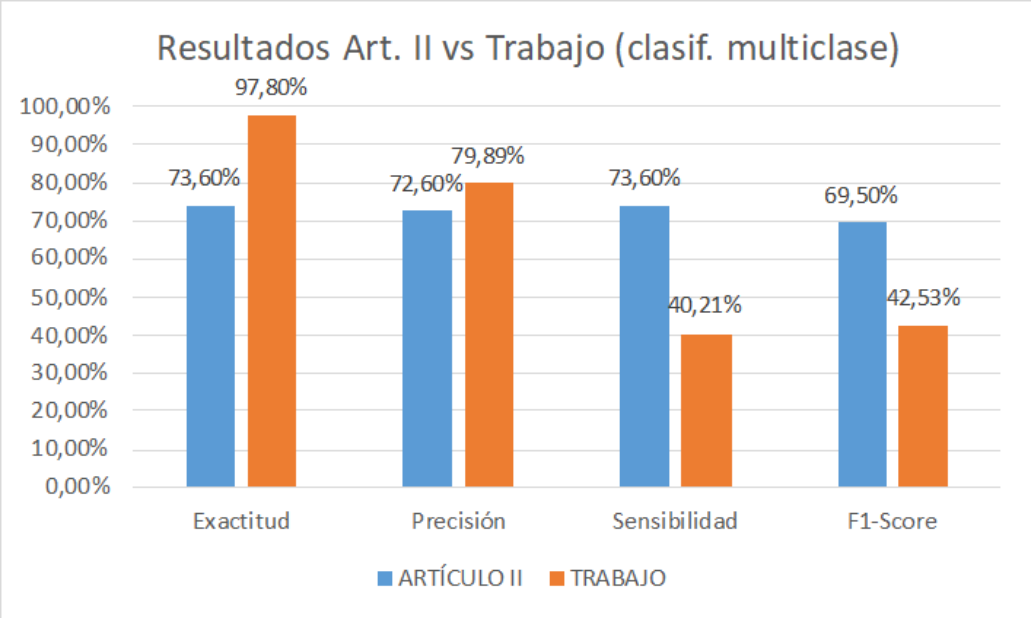


Figura 42: Comparación entre los resultados obtenidos en el presente trabajo con los expuestos en el Artículo II para el caso de clasificación multiclase.

En conclusión, se constata de los resultados y comparaciones realizadas que la base de datos no resulta nada fácil para el clasificador usado, quedando patente este hecho sobre todo a la hora de realizar clasificación multiclase. Sí que es cierto que, en condiciones normales, la gran mayoría de tráfico monitorizado es benigno (con el cual el *random forest* no tiene problemas a la hora de clasificarlo), pero para resultar eficaz en cualquier tipo de condiciones debe clasificar hasta ataques menos comunes, punto en el que el modelo programado flaquea.

En el capítulo siguiente se intenta resolver este problema o, en su defecto, aliviarlo. Se aplicará la base de datos UNSW-NB15 a un modelo de aprendizaje profundo, con objetivo de ver si se mejora el desempeño mostrado por el modelo *random forest* y establecer una comparativa directa entre los resultados obtenidos con el modelo de *machine learning* y el de *deep learning*.

5 DEEP LEARNING PARA SISTEMAS DE DETECCIÓN DE INTRUSOS

Aunque este término ya fue presentado en capítulos anteriores, en este apartado se profundiza mucho más en la relación existente entre el aprendizaje profundo y la ciberseguridad (más concretamente con los sistemas de detección de intrusos). Se exponen de forma más clara y concisa las diferencias existentes entre el aprendizaje máquina y el aprendizaje profundo, del mismo modo que las ventajas existentes de los algoritmos de *deep learning* sobre otros métodos de *machine learning* y que lo pueden hacer más atractivo para tareas de detección de intrusos. En la segunda mitad de este capítulo, se desarrollarán dos algoritmos de aprendizaje profundo que serán entrenados con la UNSW-NB15: un autocodificador variacional condicional (modelo sobre el que se centra el desarrollo de este trabajo) y un perceptrón multicapa, para comparar las prestaciones de este modelo de aprendizaje profundo con el CVAE. Se cierra esta sección enumerando las conclusiones más importantes arrojadas de la comparativa entre los resultados del CVAE a la hora de clasificar la UNSW-NB15 y el resto de modelos desarrollados (árboles de decisión y MLP), intentando explicar el porqué de cada una de ellas.

5.1 Deep learning y ciberseguridad

El aprendizaje profundo cada vez tiene mayor presencia en el campo de la ciberseguridad. De hecho, muchas opiniones coinciden en que la tendencia que siguen los avances en este sector (y en otras aplicaciones) va cada vez más orientada a soluciones relacionadas con el *deep learning*. Relacionadas con la seguridad, las soluciones empleadas utilizando aprendizaje profundo incluyen múltiples redes profundas que se usan para mejorar el rendimiento de los sistemas de detección de intrusos. Cada vez resultan más numerosos los estudios basados en técnicas de aprendizaje profundo y métodos de aprendizaje no supervisados [42].

En comparación con los modelos de aprendizaje máquina los algoritmos de aprendizaje profundo tienen capacidades de generalización más potentes, aprendiendo las características directamente de los datos en bruto; además, son totalmente independientes de los procesos de extracción de características [80]. Estos modelos son capaces de establecer relaciones entre características y combinarlas de forma totalmente autónoma, sin necesidad de pedirle al algoritmo de forma explícita que lo haga. Esta característica diferencial permite a los investigadores ahorrar bastante tiempo e incluso descubrir correlaciones entre atributos que los humanos no percibirían. Pero ésta no es solo la única diferencia existente entre ambas.

Usualmente, la complejidad computacional de los modelos de aprendizaje profundo es mucho más superior que la de los modelos de aprendizaje máquina. La cantidad de parámetros existentes en un algoritmo de DL puede ser enorme, lo que implica también una mayor cantidad de tiempo empleado en la fase de entrenamiento a diferencia de lo que pasa con un modelo de aprendizaje máquina. Además, el

número de operaciones matriciales realizado en los modelos de DL es grande, por lo que se aprovecha la potencia de las unidades de procesamiento gráfico (GPU) que aceleran en gran medida el cálculo de estas operaciones [4]. Por estos motivos, los modelos de DL normalmente necesitan de estaciones de trabajo o equipos de altas capacidades para ser ejecutados, mientras que otros algoritmos de ML pueden ser ejecutados en equipos de propósito general [81].

En lo referente a la hora de desarrollar un modelo de ML, el enfoque es bastante similar en cuanto a los pasos a seguir, no existe diferencia en ese aspecto, ya que el DL es un subconjunto del ML. Sin embargo, en DL sí que resulta ligeramente distinta la manera de ejecutar dichos pasos del desarrollo de un modelo de aprendizaje: en muchos algoritmos de ML el desarrollo de todo el modelo se divide en fases que se resuelven una a una, descomponiendo el problema general en subproblemas resueltos de forma independiente; mientras tanto, en los métodos de DL el problema se resuelve de extremo a extremo.

Por último, un factor muy relevante a la hora de establecer diferencias en la comparativa entre el *deep learning* y otros métodos de *machine learning* reside en la interpretabilidad de los resultados. Uno de los aspectos más discutidos del aprendizaje profundo es el hecho de que no se sabe realmente la manera en la que una red neuronal llega a una solución concreta. Al igual que el cerebro humano, el razonamiento de las redes neuronales se debe al comportamiento de neuronas simuladas, dispuestas en decenas o cientos de capas interconectadas entre sí. Juntas forman una compleja red donde las entradas se envían de una capa a otra hasta que se obtiene la salida del modelo. Aunque las redes neuronales producen grandes resultados, el hecho de no saber de forma concreta el modus operandi llevado a cabo por el modelo para obtener un resultado concreto hace que sea difícil predecir cuándo pueden producirse fallos [82]. Por el contrario, un algoritmo de aprendizaje máquina genera reglas explícitas de decisión para producir la salida final, por lo que resulta mucho más intuitivo encontrar el porqué de esa salida y localizar las posibles fallas existentes.

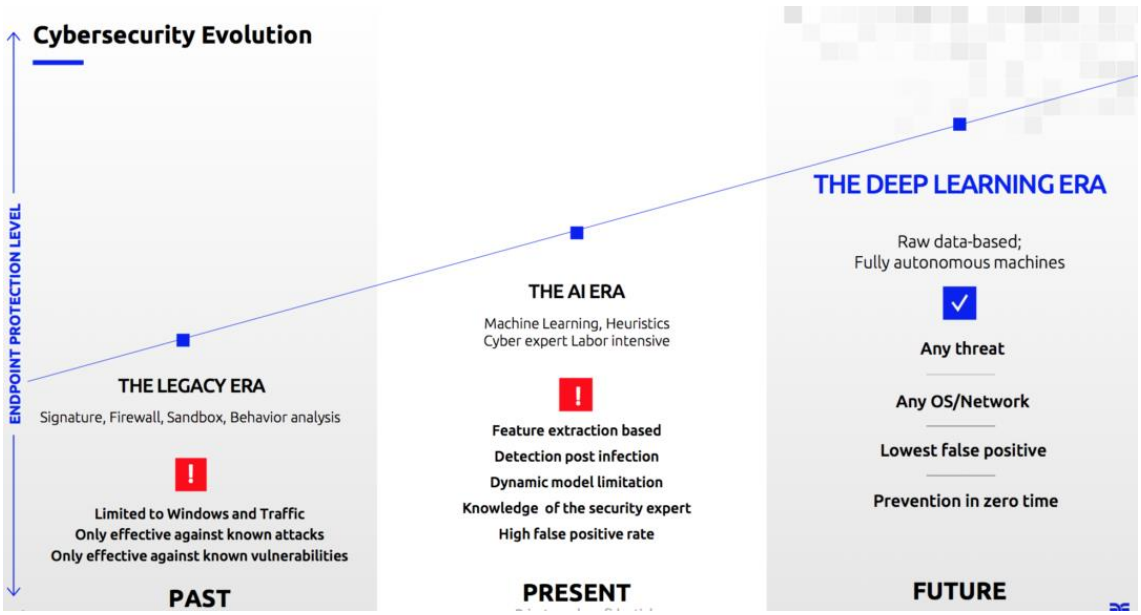


Figura 43: Gráfico que ilustra las distintas tendencias a lo largo del tiempo acerca de las soluciones empleadas en aplicaciones de ciberseguridad. Fuente: informationage.com

Después de esta breve introducción para poner en valor el *deep learning* dentro del *machine learning*, a la vez que se enfatizan sus particularidades, es necesario hablar de los distintos modelos que se engloban en esta rama de la inteligencia artificial. Existe una gran variedad de modelos distintos en esta subcategoría del ML, y la siguiente sección recoge la explicación de algunas de las más importantes.

5.2 Autocodificador variacional (VAE)

Al igual que las redes de Boltzmann restringidas, los autocodificadores variacionales también se engloban tanto dentro de las redes generativas profundas. Algunos de los problemas presentados por este tipo de redes históricamente están relacionados con la complejidad computacional de los procesos de inferencia usados en ellos, o con aquellos algoritmos que han requerido de fuertes suposiciones sobre la estructura de los datos para funcionar. Actualmente, se ha avanzado mucho sobre estos problemas, mejorando la velocidad del entrenamiento y resolviendo el problema de las suposiciones sobre la estructura de los datos. Concretamente, el VAE es una de las técnicas desarrolladas bajo el paraguas de estas mejoras, convirtiéndose en uno de los métodos más populares y prometedores dentro de los modelos generativos profundos [52].

Los autocodificadores variacionales fueron presentados en 2014 por Diederik Kingma y Max Welling y se diferencian del resto de la familia de los autocodificadores por dos motivos en particular [40]:

- Son autocodificadores probabilísticos, esto es, sus salidas están en parte determinadas por el azar, como se explicará posteriormente.
- Son autocodificadores generativos: pueden generar nuevas muestras que parecen haber sido sacadas del conjunto de entrenamiento.

De forma más precisa, se pueden definir los autocodificadores variacionales como un algoritmo de aprendizaje profundo no supervisado basado en aprender representaciones latentes de los datos de entrada, con el fin de comprender la construcción de este conjunto y poder generar muestras sintéticas a partir de las representaciones aprendidas. Desde un punto de vista más teórico, un VAE es un modelo capaz de generar datos X a partir de una distribución desconocida $P(X)$ [52].

Actualmente, las aplicaciones en las que este algoritmo está involucrado son numerosas y de diversa índole; en efecto, si se realiza una búsqueda del término “Variational autoencoder” en la base de datos del IEEE en los últimos 5 años se cuentan hasta 570 entradas. Entre todas ellas destacan los avances en el ámbito de la acústica y el sonido, utilizándose un VAE para generar piezas musicales nuevas a partir de las características de estilo de un compositor en concreto o de un estilo musical [83], o bien como un método de separación de fuentes [84]. En el sector de la medicina también ha tenido presencia, usándose para la detección de lesiones cerebrales [85] o para identificar sonidos cardíacos anormales que podrían ser indicativo de patologías de corazón [86].

5.2.1 Estructura y funcionamiento de un VAE

Un autocodificador variacional es una red neuronal formada por tres componentes esencialmente: el codificador, el espacio latente y el decodificador. El codificador es una red neuronal compuesta de una o más capas de neuronas que transforma las entradas que se le proporcionan en una representación interna (espacio latente) de dimensión muy inferior a las variables iniciales de entrada. Así se le obliga a realizar una compresión eficiente de los datos iniciales, extrayendo de éstos las características más importantes [87]. Este espacio latente está formado por variables latentes; variables que no son observadas de forma explícita, sino que son inferidas a partir de la observación de datos (por ejemplo, se podría formar un vector de variables latentes a partir de todos los rostros que una persona ha visualizado: [ojos verdes, pómulos prominentes, barbilla afilada…]). A partir de este vector de variables latentes se podrían generar otros datos nuevos tomando muestras de él [52].

Seguidamente, esta representación interna contenida en el espacio latente se usa como entrada del

decodificador (otra red neuronal) para recuperar de la forma más exacta posible el dato inicial. Al almacenar en el espacio latente las características más relevantes del dato de entrada para poder efectuar su reconstrucción, podemos usar la salida del codificador no sólo para reproducir el dato inicial, si no para poder generar muestras similares a él [87].

De forma general, las dos redes neuronales que forman parte de la estructura del autocodificador variacional (el codificador y el decodificador) se entrenan simultáneamente como una unidad con la finalidad de ajustar correctamente los comportamientos de una con respecto a la otra y viceversa, y se usa la función de pérdida como mecanismo para dirigir el entrenamiento y conseguir que el error existente entre el dato original y el reproducido por el modelo se reduzca paulatinamente. Esta función de pérdida también suele ser conocida como pérdida de reproducción (debido a la finalidad de intentar reproducir a través de la representación latente el dato de entrada), y asegura que se penalizan las configuraciones de la red que producen salidas distintas de la entrada introducida.

A continuación, se presentan los tres elementos principales que conforman la estructura de un VAE de forma más detallada [52], [88]:

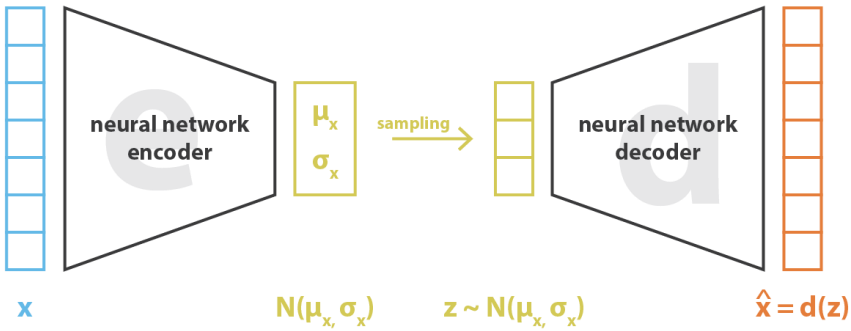


Figura 44: Estructura de un autocodificador variacional. Fuente:towardsdatascience.com

- Codificador: Se le pasan datos de entrada y produce una representación más densa y reducida (codificación) que se recoge en un espacio latente. Esta representación consiste en información de alto nivel sobre los datos de entrada. Resulta fundamental para obtener buenos resultados en la fase de decodificación, ya que debe crear variables latentes ricas en información.

- Espacio latente: Contiene la información suficiente para que el decodificador la procese adecuadamente y devuelva el formato de salida esperado. La salida del espacio latente son directamente los datos de entrada del decodificador.

- Decodificador: A partir de las variables latentes generadas por el codificador y que están almacenadas en el espacio latente, el decodificador reconstruye las entradas.

En esencia, la estructura descrita en los párrafos anteriores es general para toda la familia de autocodificadores; sin embargo, sí que existe una característica que diferencia los autocodificadores variacionales de un autocodificador general en cuanto a estructura, y que resulta ser el elemento clave para la realización de tareas generativas del VAE. El problema fundamental con los autocodificadores

clásicos a la hora de generar nuevas muestras es que el espacio latente en el cual el codificador transforma las entradas normalmente no es continuo (tiene discontinuidades) o no permite una fácil interpolación. Entonces, a la hora de tomar aleatoriamente una muestra de este espacio latente para generar nuevas instancias, si la muestra se escoge de una de estas discontinuidades el decodificador dará una salida poco realista, ya que el decodificador no sabe cómo lidiar con esa región del espacio latente [89].

Esto no ocurre en un VAE, cuyo espacio latente es continuo, permitiéndose así el muestreo aleatorio o la interpolación sin que haya problema. En lugar de hacer que el codificador genere un vector prefijado de tamaño *n* (caso del AE)[15], devuelve como salida dos vectores de tamaño *n*: un vector de medias μ y otro vector de desviaciones estándar, σ. Conjuntamente, ambos vectores forman un vector de variables aleatorias descritas por medio de distribuciones normales. De este vector de distribuciones normales se muestrea de forma aleatoria el vector que se pasará al decodificador. Esta generación estocástica hace que, para la misma muestra de entrada, aunque el vector de distribuciones normales sea el mismo la codificación de la muestra sea distinta debido al muestreo aleatorio de este vector de distribuciones normales [89]. Así se pueden generar nuevas muestras sintéticas.

5.2.2 Fundamentos matemáticos de un VAE

Después de haber ofrecido una explicación a alto nivel de la estructura y comportamiento de un autocodificador variacional, en las siguientes líneas se desgrana su funcionamiento desde un punto de vista puramente matemático.

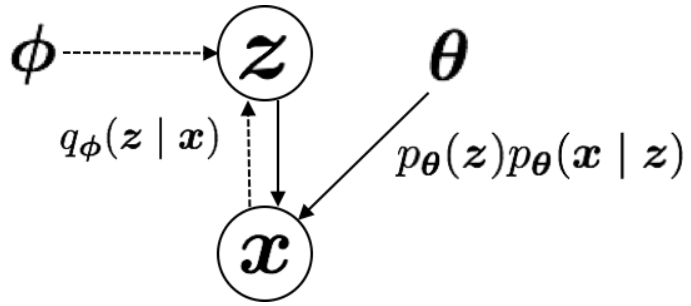


Figura 45: Modelo gráfico del problema. Las líneas continuas denotan el modelo generativo $p_\theta(z)p_\theta(x|z)$, las líneas discontinuas denotan la aproximación $q_\phi(x|z)$ a la probabilidad intratable $p_\theta(x|z)$. Los parámetros variacionales ϕ se aprenden de forma conjunta con los parámetros generativos θ del modelo. Fuente: renom.jp

Se comienza definiendo en primer lugar cierta terminología que será recurrente a lo largo de la explicación. Para facilitar la comprensión de qué es cada uno de los elementos que se describirán a continuación, se acompaña la descripción de algunos términos de una analogía con un ejemplo real [90]:

- $\boldsymbol{x}$: son los datos de entrada del modelo; por ejemplo, animales.
- $\boldsymbol{z}$: son las variables latentes, que se corresponderían con la imaginación humana.
- $p_\theta(\boldsymbol{x})$: distribución de probabilidad de los datos de entrada. Esto se podría asemejar con todas las especies de animales existentes. θ se corresponde con los parámetros que definen dicha distribución de probabilidad.
- $p_\theta(\boldsymbol{z})$: distribución de probabilidad de la variable latente. Siguiendo con las analogías, $p_\theta(\boldsymbol{z})$ se correspondería con el cerebro, fuente de la imaginación humana.

[15] *n es el número de dimensiones del espacio latente.*

- $p_\theta(\boldsymbol{x}|\boldsymbol{z})$: es la distribución de los datos generados dada la variable latente, es decir, el animal generado a partir de la imaginación.
- $q_\phi(\boldsymbol{x}|\boldsymbol{z})$: distribución de probabilidad que servirá para intentar inferir $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, la cual resulta intratable. ϕ se corresponde con los parámetros que definen dicha distribución de probabilidad.

En la Figura 45 se observa la premisa inicial sobre la que se sustenta el funcionamiento de un VAE. Desde una perspectiva probabilística, un autocodificador variacional es un modelo compuesto por los datos de entrada $\boldsymbol{x}$ y las variables latentes $\boldsymbol{z}$. La probabilidad conjunta del modelo puede escribirse como:

$$p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p_\theta(\boldsymbol{z}) \tag{18}$$

Las variables latentes se extraen de la probabilidad a priori $p_\theta(\boldsymbol{z})$, mientras que los datos $\boldsymbol{x}$ tienen una probabilidad condicionada por las variables latentes $\boldsymbol{z}$, $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. El objetivo es inferir características de las variables latentes $\boldsymbol{z}$ a partir de los datos observados $\boldsymbol{x}$ o, dicho de otra forma, calcular la probabilidad a posteriori $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Según el Teorema de Bayes[16] [91]:

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{x})} \tag{19}$$

Prosiguiendo con el desarrollo, al marginalizar las variables latentes $\boldsymbol{z}$ se puede escribir el denominador $p_\theta(\boldsymbol{x})$ de la siguiente forma:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}, \boldsymbol{z})\, dz \tag{20}$$

Esta probabilidad $p_\theta(\boldsymbol{x})$ es dependiente de los pesos del modelo, θ. La finalidad es encontrar un conjunto de pesos θ que maximice $p_\theta(\boldsymbol{x})$ para conseguir un buen modelo generativo [52]. Desafortunadamente, de forma bastante común esta integral resulta intratable a causa de que requiere de una cantidad de tiempo exponencial para calcularse; además, necesita ser evaluada en todas las variables latentes. Por ende, $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ resulta imposible de calcular también. Además, otro problema radica en el tamaño de los conjuntos de datos tratados. En ocasiones, estos pueden ser muy grandes y dar lugar a problemas de memoria [92]. La solución al primer problema comentado pasa por aproximar la probabilidad a posteriori $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, la cual es intratable, por una distribución estimada $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, más simple y tratable. Dicho de otra forma, en un autocodificador variacional se añade otra red neuronal con una distribución $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ para intentar inferir la distribución real $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Esta red no es otra cosa que el codificador que ya se mencionó en líneas anteriores [52]. Para llevar a cabo esta aproximación se utiliza un método denominado inferencia variacional (VI), desarrollado de forma más completa en la sección siguiente.

Para finalizar esta especie de introducción y una vez habiendo definido toda la terminología necesaria, se puede hablar de la estructura del modelo también en lenguaje probabilístico. La red neuronal que conforma el codificador recibe como entrada unos datos $\boldsymbol{x}$ y devuelve como salida una representación latente $\boldsymbol{z}$ (produce una distribución sobre todos los posibles valores de $\boldsymbol{z}$ a partir de los cuales se podría haber generado $\boldsymbol{x}$). Es por esto que el codificador se puede expresar como $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, donde la variable ϕ representa sus pesos y sesgos. Por otro lado, la red neuronal que compone el decodificador

[16] *El Teorema de Bayes afirma que:* $p(A|B) = \frac{p(B|A)p(A)}{p(B)} \rightarrow prob.\,a\ posteriori = \frac{verosimilitud * p.priori}{p.marginal\ (o\ evidencia)}$

toma como entrada la representación latente $\boldsymbol{z}$ y reconstruye la entrada $\boldsymbol{x}$ siguiendo una distribución de probabilidad; por ello se denota el decodificador como $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, siendo θ sus pesos y sesgos [92].

Por último, para medir el rendimiento de nuestro modelo y cuantificar si éste está aprendiendo de forma correcta o no, se utiliza la función de pérdida como se comentó anteriormente. Esta función constará de dos medidas principales: en primer lugar, se debe computar la capacidad del codificador para construir variables latentes que reproduzcan fielmente los datos de entrada $\boldsymbol{x}$ siguiendo una distribución $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ (este término se conoce como pérdida latente) y por otro lado, se debe calcular el desempeño del decodificador para reconstruir las variables de entrada a partir de las variables sitas en el espacio latente, siguiendo una distribución $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ (este término se conoce como pérdida generativa) [93]. En apartados posteriores se tratará profusamente esta función de pérdida y se detallarán sus elementos.

I. Inferencia variacional

La inferencia variacional es una de las técnicas más usadas dentro del campo de la inferencia bayesiana [94]. La finalidad de la inferencia variacional es aproximar una función densidad de probabilidad compleja a otra que sea tratable mediante un problema de optimización. Además, resulta un método más rápido y fácilmente escalable a gran cantidad de datos comparado con otras alternativas, como el método de Montecarlo basado en cadenas de Markov (MCMC), más enfocado a datasets de menor tamaño. Al final, ambos métodos resultan ser enfoques distintos que vienen a resolver el mismo problema: la inferencia variacional mediante la resolución de un problema de optimización y el MCMC muestreando una cadena de Markov [95].

En el caso del autocodificador variacional, se tiene una distribución de probabilidad $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ intratable. El método de inferencia variacional tratará de encontrar la mejor aproximación a la distribución de probabilidad $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ de entre una familia de distribuciones Q_ϕ tratable, a través de la resolución de un problema de optimización. Una vez resuelto este problema, $q_\phi^*(\cdot)$ resulta ser la distribución de probabilidad dentro de Q_ϕ más cercana a $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ [22]. El parámetro ϕ depende de la familia de distribuciones: si Q_ϕ es una gaussiana, ϕ se corresponde con la media y la varianza de las variables latentes para cada uno de los datos de entrada [52]:

$$\phi = (\mu, \sigma^2)$$

Para encontrar la distribución de probabilidad que más se acerca a $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, se debe medir el parecido o similitud existente entre cada $q_\phi(\boldsymbol{z}|\boldsymbol{x}) \in Q_\phi$ y la propia $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Para ello se utiliza la divergencia de Kullback-Leibler, que mide la diferencia existente entre dos distribuciones de probabilidad o, dicho de otra forma, el número de bits de información perdida si se usa $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ para representar $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ [23]. Queda patente de esta última definición de la divergencia KL que el objetivo último es minimizar la diferencia entre las dos distribuciones de probabilidad, puesto que así la pérdida de información será la mínima posible. El problema de optimización a resolver es el siguiente [95]:

$$q_\phi^*(\boldsymbol{z}|\boldsymbol{x}) = \underset{q_\phi(\boldsymbol{z}|\boldsymbol{x}) \in Q_\phi}{\arg\min} \; KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})) \tag{21}$$

donde la divergencia KL se puede desarrollar como:

$$KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \sum_{z\in \mathrm{Z}} q_\phi(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p_\theta(\boldsymbol{z}|\boldsymbol{x})} \quad (22)$$

La divergencia de Kullback-Leibler es mayor o igual a cero, al medir una diferencia entre dos distribuciones de probabilidad; además es asimétrica, esto es: $KL(p||q) \neq KL(q||p)$. Después de cierta matemática y del empleo de las propiedades de los logaritmos, la expresión (22) se puede reformular como[17] [95]:

$$KL\left(q_\phi(\boldsymbol{z}|\boldsymbol{x})\middle|p_\theta(\mathbf{z}|\mathbf{x})\right) = \mathbb{E}\left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right] - \mathbb{E}\left[\log p_\theta(\mathbf{z},\mathbf{x})\right] + \log p_\theta(\mathbf{x}) \quad (23)$$

Observando (23) se visualiza la dependencia de esta expresión con $p_\theta(\boldsymbol{x})$, lo cual directamente convierte esta divergencia KL en intratable (al serlo $p_\theta(\boldsymbol{x})$, como se comentó en líneas previas). Al no poder computarse directamente (23), es necesario buscar alternativas para resolver la optimización. Concretamente, se utiliza como sustituta de (23) una función objetivo denominada como ELBO (Evidence Lower Bound Objective) [95]:

$$\mathrm{ELBO}(q) = \mathbb{E}\left[\log p_\theta(\mathbf{z},\mathbf{x})\right] - \mathbb{E}\left[\log q_\phi(\mathbf{z}|\mathbf{x})\right] \quad (24)$$

Se reescribe el ELBO como una suma de la probabilidad logarítmica esperada de los datos y la divergencia KL entre $p_\theta(\mathbf{z})$ y $q_\phi(\boldsymbol{z}|\boldsymbol{x})$:

$$\begin{aligned}\mathrm{ELBO}(q) &= \mathbb{E}\left[\log p_\theta(\mathbf{z})\right] + \mathbb{E}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - \mathbb{E}\left[\log q_\phi(\mathbf{z}|\mathbf{x})\right] \\ &= \mathbb{E}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})|p_\theta(\mathbf{z}))\end{aligned} \quad (25)$$

El primer término de (25) resulta ser el error de reconstrucción, mientras que el segundo es un término de regularización formado por la divergencia KL negativa entre la distribución que aproxima $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ y la probabilidad a priori $p_\theta(\mathbf{z})$. Normalmente, esta probabilidad a priori suele ser una distribución normal, por comodidad y facilidad a la hora de operar.

Una propiedad interesante del ELBO es que es el límite inferior de la evidencia:

$$\log p(\boldsymbol{x}) \geq \mathrm{ELBO}(q)$$

Pudiéndose reescribir entonces $\log p(\boldsymbol{x})$ de la siguiente forma [95]:

$$\log p(\boldsymbol{x}) = \mathrm{ELBO}(q) + KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})|p_\theta(\boldsymbol{z})) \quad (26)$$

Centrando la atención en esta última expresión y recordando que la divergencia KL es no negativa, se puede deducir que maximizando el ELBO se minimiza la divergencia KL y, en consecuencia, se mejora la evidencia $p_\theta(\mathbf{x})$. En otras palabras, maximizar el ELBO provoca que los datos de entrada se describan mucho mejor, ya que se mejora $p_\theta(\mathbf{x})$ (se tiene pues, un mejor modelo generativo) [96].

II. Optimización de la función de pérdida

Como ya se ha expuesto en varias ocasiones a lo largo de este capítulo, la función de pérdida permite cuantificar el error existente entre los datos deseados y los datos de salida arrojados por el modelo. En el apartado anterior, se llegó a la conclusión de que para el autocodificador variacional, la función de

[17] *El desarrollo completo de la descomposición se puede consultar en [24].*

pérdida a optimizar era el ELBO y el objetivo era maximizarlo.

Sin embargo, lo usual en las funciones de pérdida resulta ser buscar la minimización de dicho término, puesto que la finalidad de dicha función de pérdida es minimizar el error existente entre la salida y la entrada. Entonces, respetando este criterio la función de pérdida de un VAE quedaría representada por la siguiente expresión [52]:

$$Loss\ function = -ELBO$$
$$= \underbrace{-\mathbb{E}[\log p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{PÉRDIDA GENERATIVA}} + \underbrace{KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})|p_\theta(\mathbf{z}))}_{\text{PÉRDIDA LATENTE}} \quad (27)$$

Como se puede observar, la expresión (27) se compone de dos términos: la pérdida latente y la pérdida generativa. Aunque previamente ya se proporcionó una definición general del significado de ambos, se vuelven a describir a continuación desde un punto de vista más concreto:

- **Pérdida latente o término de regularización:** Se mide a través de la divergencia KL y trata de asegurar que las variables contenidas en el espacio latente describen de forma eficiente los datos de entrada [97].

- **Pérdida generativa o error de reconstrucción:** Intenta medir la eficacia conjunta del codificador y el decodificador con respecto a los datos iniciales, esto es, cuantificar cómo de bien reconstruye el modelo las entradas a partir de las salidas obtenidas. Si este término fuese nulo, el VAE podría reconstruir perfectamente los datos de entrada. Sin embargo, esto provocaría un sobreajuste extremo en el modelo [98].

Para la optimización de la función de pérdida comúnmente se utilizan técnicas relacionadas con el descenso de gradientes. Estos algoritmos minimizan la función objetivo y actualizan los parámetros de la red (pesos y sesgos) de la siguiente forma:

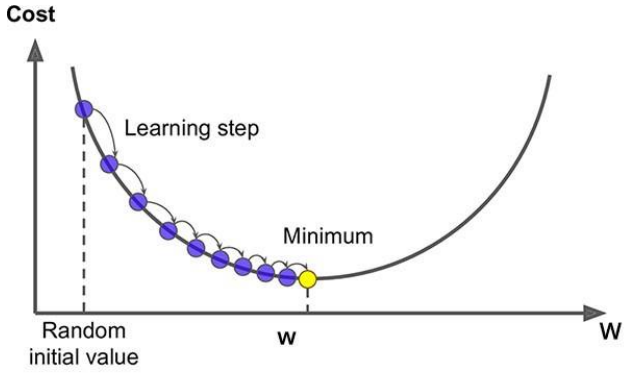


Figura 46: Representación gráfica del procedimiento a seguir por un algoritmo de descenso de gradientes. Fuente: mc.ai

De forma iterativa y partiendo de un valor aleatorio inicial, un algoritmo de descenso de gradientes va moviéndose en dirección negativa al gradiente de la función de pérdida hasta alcanzar el mínimo de la función. La "longitud del paso" con el que el algoritmo se dirige hacia este mínimo está controlado por un parámetro denominado tasa de aprendizaje. Existen muchos tipos de algoritmos de este tipo,

como pueden ser el Momentum, Adam, o Adadelta, entre otros [99]. Para poder aplicar este tipo de métodos es necesario que toda la red sea diferenciable para poder calcular las derivadas sin ningún problema.

Sin embargo, existe un inconveniente a la hora de aplicar esta técnica al ELBO, y que impide que la expresión sea diferenciable. El problema reside principalmente a la hora de aplicar la diferenciación al muestreo aleatorio de las distribuciones normales existentes en el espacio latente: para un proceso de muestreo aleatorio no se puede aplicar la propagación hacia atrás [88]. Aquí es donde aparece la técnica conocida como "reparametrization trick" o truco de la reparametrización. Básicamente, este método desvía el comportamiento no diferenciable de la red fuera del proceso, permitiendo de esta forma el entrenamiento y actualización de los pesos de ésta [90].

III. Truco de la reparametrización

Para sortear el obstáculo de aplicar la propagación hacia atrás en el caso de tener muestreo aleatorio en la red neuronal, se aplica este método. Detallando la expresión (8) se tiene lo siguiente [92]:

$$\mathrm{ELBO}(\boldsymbol{\phi}, \boldsymbol{\theta}) = \mathbb{E}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z})) \tag{28}$$

Como se explicaba en la sección anterior, el objetivo es optimizar y diferenciar el ELBO con respecto a los parámetros de los que depende, ϕ y θ. No obstante, el cómputo del gradiente con respecto a estas dos variables resulta ser bastante problemático por la razón ya descrita previamente. Para intentar desplazar esta operación no diferenciable fuera de la red se suele estimar el gradiente por Monte Carlo:

$$\begin{aligned} \nabla_\phi \mathbb{E}_{q_\phi(\boldsymbol{z})}[f(\boldsymbol{z})] = \mathbb{E}_{q_\phi(\boldsymbol{z})}\left[f(Z)\nabla_{q_\phi(\boldsymbol{z})}\log q_\phi(\boldsymbol{z})\right] \cong \\ \cong \frac{1}{L}\sum_{l=1}^{L} f(z)\,\nabla_{q_\phi(\boldsymbol{z}^l)}\log q_\phi(\boldsymbol{z}^l) \end{aligned} \tag{29}$$

donde $\boldsymbol{z}^l \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})$.

Para llevar a cabo esta operación, se reparametriza la variable aleatoria $Z^l \sim q_\phi(Z|X)$ usando una transformación diferenciable $g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$, con una variable de ruido $\boldsymbol{\epsilon}$:

$$\tilde{\boldsymbol{z}} = g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}),\ con\ \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon}) \tag{30}$$

Una vez realizado este cambio, se puede computar la estimación por Monte Carlo de la esperanza de $f(\boldsymbol{z})$ con respecto a $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ usando la siguiente expresión [92]:

$$\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})}[f(\boldsymbol{z})] = \mathbb{E}_{p(\boldsymbol{\epsilon})}\left[f\left(g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x}^{(i)})\right)\right] \cong \frac{1}{L}\sum_{l=1}^{L} f\left(g_\phi(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x}^{(i)})\right) \tag{31}$$

donde $\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$.

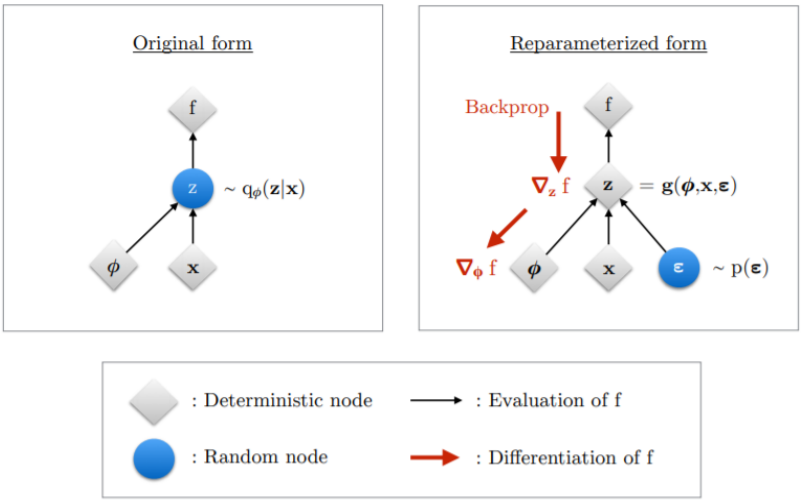


Figura 47: Ilustración de la técnica de reparametrización. Se desplaza la aleatoriedad existente en z al reparametrizar esta variable como una función determinista y diferenciable respecto de ϕ, x, y una nueva variable ϵ. Esto permite la propagación hacia atrás y el cómputo de los gradientes.

Aplicando la última expresión vista al ELBO se consigue el estimador de Bayes de variación de gradiente, más conocido como stochastic gradient variational Bayes o SGVB:

$$\text{ELBO}(\boldsymbol{\phi}, \boldsymbol{\theta};\ \boldsymbol{x}^{(i)}) = \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i,l)}) - \log q_\phi(\boldsymbol{z}^{(i,l)}|\boldsymbol{x}^{(i)}) \tag{32}$$

donde $\boldsymbol{z}^{(i,l)} = g_\phi\left(\boldsymbol{\epsilon}^{(i,l)}, \boldsymbol{x}^{(i)}\right),\ \boldsymbol{\epsilon}^{(i)} \sim p(\boldsymbol{\epsilon})$, y L el número de muestras por dato.

Resulta pertinente comentar que la anterior ecuación puede expresarse de otra forma debido a que, normalmente, la divergencia KL puede ser calculada analíticamente. Cuando sucede esto, solamente el término de error de reconstrucción es el que requiere la aplicación de la estimación del muestreo aleatorio, quedando esta versión alternativa de (32):

$$\text{ELBO}(\boldsymbol{\phi}, \boldsymbol{\theta};\ \boldsymbol{x}^{(i)}) = -KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z})) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i,l)}) \tag{33}$$

donde $\boldsymbol{z}^{(i,l)} = g_\phi\left(\boldsymbol{\epsilon}^{(i,l)}, \boldsymbol{x}^{(i)}\right),\ \boldsymbol{\epsilon}^{(i)} \sim p(\boldsymbol{\epsilon})$.

Además, también se pueden sortear los inconvenientes producidos por los grandes conjuntos de datos, construyendo un estimador basado en minibatches (miniconjuntos del set completo de datos) [92]:

$$\text{ELBO}(\boldsymbol{\phi}, \boldsymbol{\theta};\ \boldsymbol{x}^{(i)}) = \frac{N}{M}\sum_{i=1}^{M} \text{ELBO}(\boldsymbol{\phi}, \boldsymbol{\theta};\ \boldsymbol{x}^{(i)}) \tag{34}$$

siendo N el número total de datos del set y M el tamaño del minibatch.

Al aplicar el truco de la reparametrización ya se puede reescribir la esperanza respecto a $q_\phi(Z|X)$, de forma que la estimación de Monte Carlo de la esperanza resulta diferenciable con respecto a ϕ. Esto permite el entrenamiento del autocodificador variacional. Los parámetros ϕ y θ son optimizados de forma conjunta junto con la función de pérdida [52].

5.3 Desarrollo de un CVAE para un sistema de detección de intrusos

Una vez planteados los fundamentos teóricos sobre los que se sustenta el funcionamiento de un autocodificador variacional, en este apartado se ponen a prueba sus capacidades como clasificador. Entrando en más detalle, se propondrá como sistema de detección de intrusos un autocodificador variacional condicional (o CVAE, por sus siglas), tomando como referencia el artículo "*Conditional Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion Detection in IoT*", de Manuel López-Martín, Belén Carro, Antonio Sánchez-Esguevillas y Jaime Lloret. En 2017, estos autores desarrollaron de forma pionera en el campo de la detección de intrusión un CVAE dedicado a tareas de clasificación y recuperación de datos perdidos a partir de las características [100]. A diferencia del presente trabajo, estos autores utilizaron la base de datos NSL-KDD para probar el rendimiento del CVAE. El objetivo es comprobar ahora el desempeño en tareas clasificatorias de este mismo modelo con la base de datos con la que se trabaja en este documento, la UNSW-NB15. A día de hoy, no se tiene constancia de ningún artículo que proponga la utilización de un CVAE como clasificador de esta base de datos en concreto.

El CVAE está basado en los conceptos del VAE, a excepción de una diferencia: en un VAE solamente se pasan como entrada a la red neuronal las características, mientras que, en un CVAE, además de pasar dichas características, se añaden también en la red las etiquetas de clase de cada una de las instancias. Este cambio proporciona muchas ventajas al autocodificador variacional condicional frente al autocodificador variacional simple.

Para emplear un VAE como clasificador, es necesario crear tantos modelos como clases distintas tenga la base de datos utilizada, entrenándose cada modelo de forma independiente (One vs. Rest). Cada paso de entrenamiento emplea como set de datos sólo las instancias referentes a la clase aprendida. En contraposición, el CVAE necesita crear solamente un único modelo con un solo paso de entrenamiento, empleando todos los datos del set de entrenamiento independientemente de sus etiquetas de clase al insertarse éstas en el decodificador. Por este motivo un CVAE es mucho más eficiente en términos computacionales [100].

5.3.1 Autocodificador variacional condicional (CVAE)

En la figura 48 se compara la arquitectura de un VAE con la de un CVAE[18]. Volviendo de forma resumida a lo desarrollado en la sección anterior, el objetivo es obtener la distribución de probabilidad de los datos $\boldsymbol{X}$ a través del codificador y el decodificador. El codificador genera un mapeo de los datos a unas distribuciones de probabilidad intermedias $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$ que contienen las características más representativas del conjunto de datos, formándose así el espacio latente $\mathbf{Z}$. El muestreo de estas distribuciones de probabilidad intermedias genera las llamadas variables latentes.

[18] *En la figura, ID son las siglas de Intrusion Detection.*

El decodificador toma estas variables latentes como referencia para describir nuevas distribuciones de probabilidad condicionadas $p_\theta(\mathbf{x}|\mathbf{z})$, de las que se vuelven a extraer muestras. Estas muestras conforman la salida del modelo $\widehat{\boldsymbol{X}}$. Como ya se comentó previamente, el objetivo pasa por aproximar la salida del modelo $\widehat{\boldsymbol{X}}$ al conjunto de datos inicial, esto es, minimizar la diferencia entre ambas distribuciones condicionales de probabilidad, utilizando para ello la técnica de inferencia variacional ya vista [100].

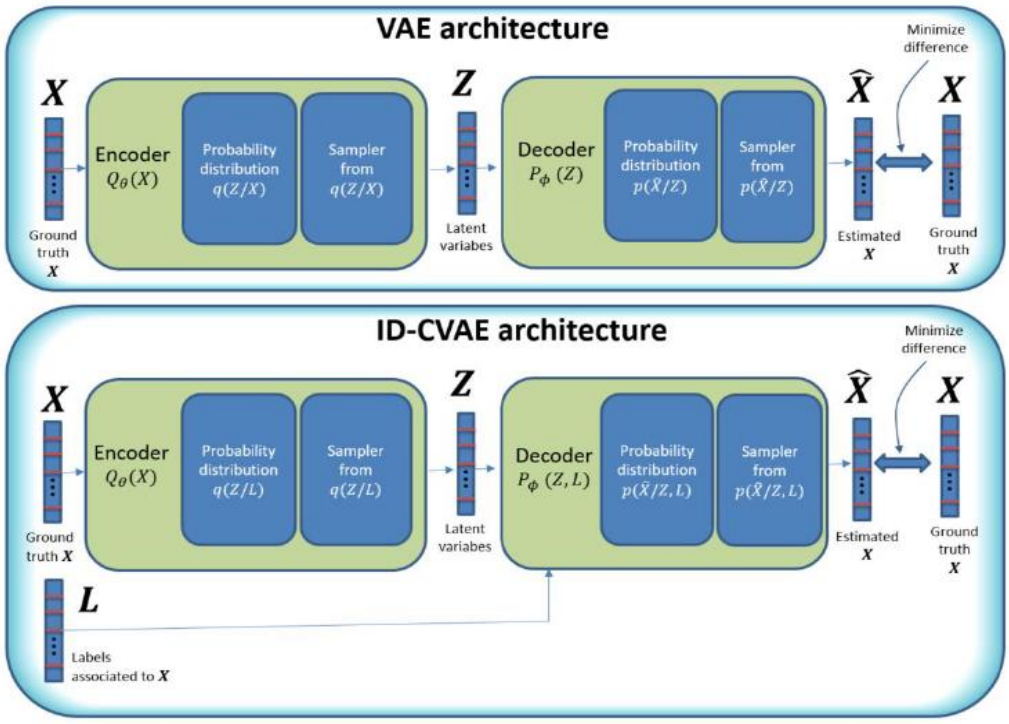


Figura 48: Comparación de la estructura del CVAE con la arquitectura del VAE. Fuente: [100].

La arquitectura del autocodificador variacional condicional resulta prácticamente análoga a la del VAE, pero las etiquetas de clase de cada una de las muestras que componen el conjunto de datos se insertan en el decodificador. Esto hace que las distribuciones de probabilidad condicionales del decodificador dependan en este caso de las variables latentes muestreadas de $\boldsymbol{Z}$ y de las etiquetas de clase $\boldsymbol{L}$**.** El resto del esquema guarda la misma estructura que la del VAE. La introducción de las etiquetas de clase en el modelo genera ciertas ventajas [100]:

- Añade información adicional al decodificador que sirve para establecer un vínculo entre las características de las instancias y las etiquetas de clase.
- Realizar tareas de clasificación efectuando un único paso de entrenamiento con todo el set de datos al completo.
- Realizar reconstrucción de características. Al mapear las características de los datos de entrada con distribuciones de probabilidad en el espacio latente, el CVAE puede recuperar características en caso de trabajar con datos de entrada incompletos (con menos características de las esperadas).

5.3.2 Procedimiento para utilizar el CVAE como clasificador

El proceso para usar el CVAE como clasificador consta de dos fases: una fase de entrenamiento y otra fase de predicción, como se observa en la siguiente figura [100]:

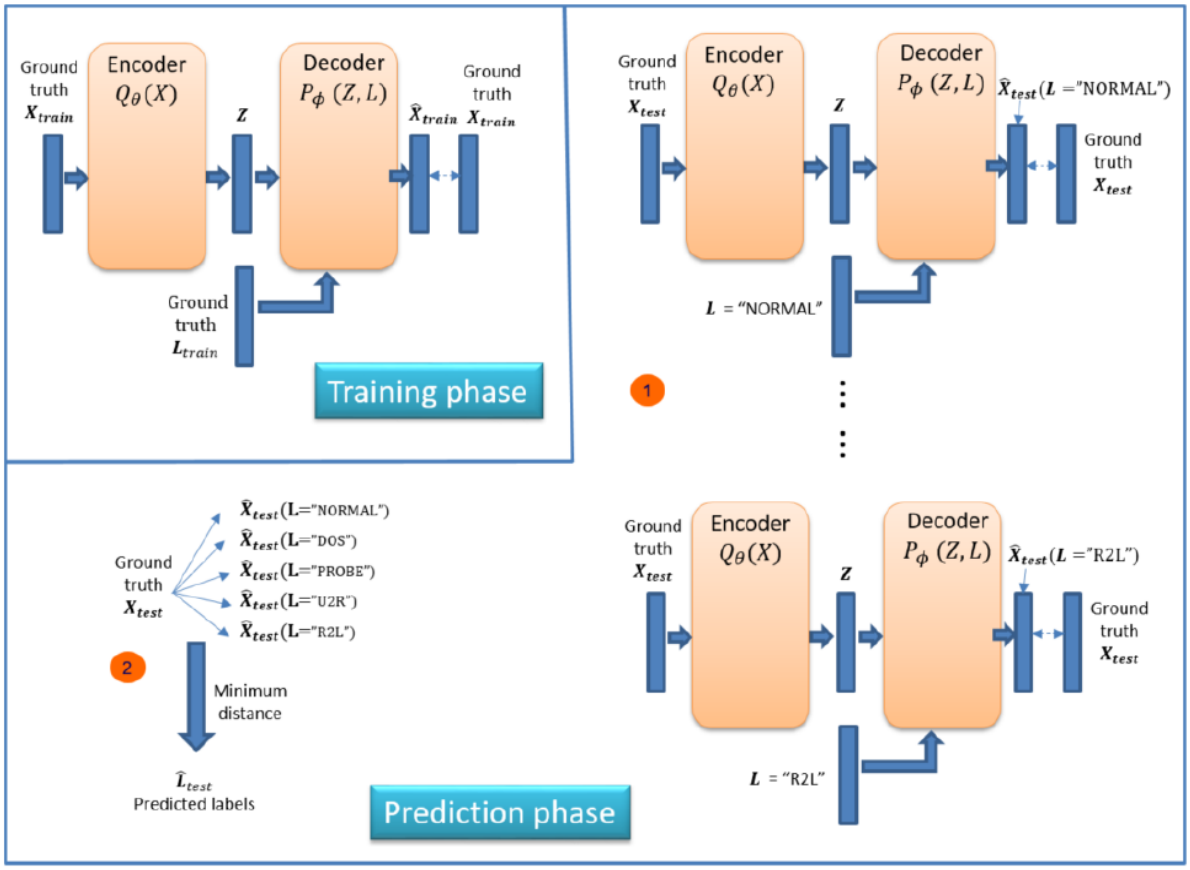


Figura 49: Estructura para emplear el CVAE como clasificador [100].

En la fase de entrenamiento, se entrena el CVAE usando el set de entrenamiento junto con las etiquetas asociadas a cada una de las instancias que componen el conjunto. El objetivo de este primer paso es minimizar la diferencia existente entre el conjunto de instancias de entrada y el conjunto recuperado a la salida: $\boldsymbol{X_{train}}$ vs $\boldsymbol{\hat{X}_{train}}$.

La fase de predicción se subdivide a su vez en dos pasos: en el primer paso, se aplica el modelo entrenado previamente a cada muestra del conjunto de datos de test $\boldsymbol{X_{test}}$ junto con un vector de etiquetas $\boldsymbol{L}$ que contiene un solo valor, para obtener a la salida una muestra reconstruida $\boldsymbol{\hat{X}_{test}}$. La idea es introducir en $\boldsymbol{L}$ solamente una única clase, y aplicarla a $\boldsymbol{X_{test}}$. Este paso se deberá ejecutar tantas veces como clases distintas haya en el conjunto de datos, cambiando cada vez la etiqueta que va en el vector $\boldsymbol{L}$. Finalmente, se tiene un conjunto de muestras reconstruidas para una misma instancia del conjunto de datos, cada una de ellas asociada a una clase distinta. En el segundo paso de esta fase de predicción, se calcula la distancia entre la muestra original y las muestras reconstruidas, eligiendo para cada muestra como etiqueta predecida aquella con la menor distancia con respecto a la etiqueta verdadera. La medida de distancia usada en este caso es la Euclídea [100].

En resumen, en el CVAE el proceso de clasificación requiere de un solo paso de entrenamiento, y tantos pasos de testeo como distintas clases haya presentes en el set de datos. El paso de entrenamiento es el que requiere de una mayor cantidad de tiempo y recursos computacionales, mientras que los

distintos pasos de testeo son mucho más rápidos y computacionalmente menos costosos. En contraposición, si se usase un autocodificador variacional para tareas clasificatorias, se requerirían tantos pasos de entrenamiento y test como etiquetas distintas tuviese el conjunto de datos.

Una vez explicadas las diferencias existentes entre un VAE y un CVAE y el proceso que sigue este último a la hora de realizar clasificación de muestras, en los siguientes apartados se muestra el desempeño de este modelo a la hora de clasificar las distintas clases de la base de datos elegida para desarrollar el presente trabajo: la UNSW-NB15.

5.3.3 Desarrollo del modelo y resultados

Para el desarrollo del autocodificador variacional condicional se parte de un modelo ya desarrollado por un usuario de GitHub, que replica la estructura del CVAE descrito anteriormente[19]. Como ya se comentó anteriormente, la base de datos original de este código era la NSL-KDD, mientras que en este trabajo se está utilizando la UNSW-NB15. Por tanto, hay que realizar las modificaciones pertinentes en el modelo para poder utilizar la otra base de datos.

Después de revisar el modelo y corregir algún detalle para replicar con exactitud lo propuesto en [100], se procede a ejecutar el código antes de ser modificado para comprobar si es capaz de alcanzar los resultados propuestos en el artículo con la NSL-KDD. La siguiente figura muestra la comparativa de métricas entre el paper original y la ejecución del código:

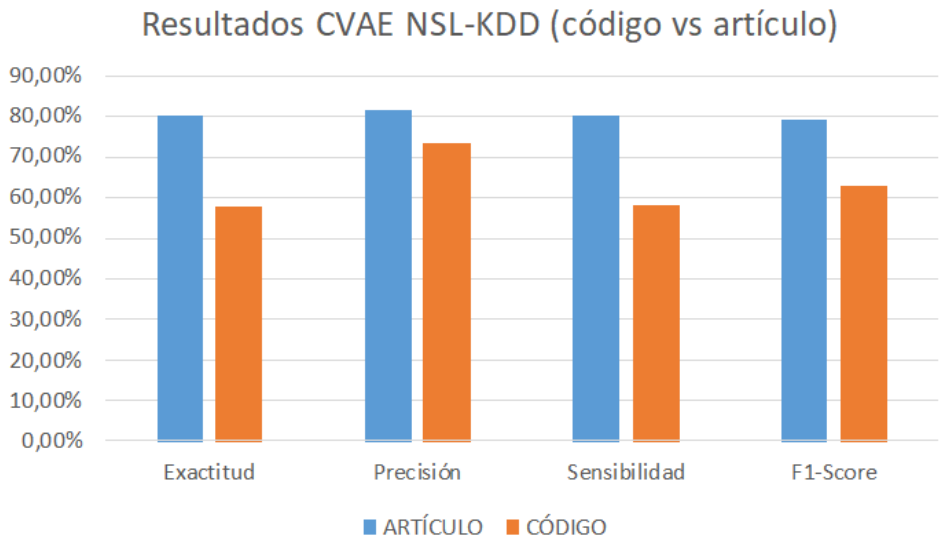


Figura 50: Comparación de los resultados obtenidos para la NSL-KDD entre el artículo [100] y el código de GitHub. Se ha seleccionado "weighted avg" en las métricas del código de GitHub para realizar la comparativa (es la que se usa en el artículo).

La disparidad entre ambos resultados (artículo y código) queda bastante patente. Desarrollando el modelo con los datos mostrados en el artículo, las métricas quedan bastante lejos de las conseguidas por los autores del documento. No se ofrecen detalles sobre el procedimiento seguido para el desarrollo del modelo en lo que a código se refiere, por lo que resulta difícil replicar fielmente el comportamiento del modelo para llegar a los resultados descritos en el paper. Solamente se explica de manera cualitativa el comportamiento del algoritmo. Esta diferencia notable de resultados lleva a

[19] *Se puede encontrar el código original en https://github.com/imoken1122/Intrusion-Detection-CVAE*

pensar que la aplicación de la base de datos UNSW-NB15 a este modelo no arrojará unos resultados muy elevados, teniendo en cuenta que incluso los resultados originales del artículo no superan el 90% para una base de datos a priori con menos dificultad que la utilizada en este trabajo.

Teniendo presente esta diferencia de resultados, se pasa a realizar la adaptación de este modelo de aprendizaje profundo para ser usado en la UNSW-NB15. La estructura de archivos que componen el modelo completo consta de 4 códigos que realizan las siguientes funcionalidades[20]:

- *preprocessing.py:* se realiza el preprocesado y preparación de los conjuntos de entrenamiento y test de la base de datos UNSW-NB15 para su posterior clasificación binaria o multiclase por parte del CVAE.
- *model.py:* se define la estructura de capas neuronales del CVAE por las que pasan las instancias. Este modelo es invocado en *cvae.py* y *valuate.py*, para realizar el entrenamiento y clasificación de las muestras, respectivamente.
- *cvae.py*: se realiza el entrenamiento (configuración de pesos y sesgos del modelo) y cálculo de la función de pérdida para los conjuntos de entrenamiento y test para un número de iteraciones determinado. Los conjuntos de datos utilizados son los generados en el fichero *preprocessing.py*, pertenecientes a la base de datos UNSW-NB15. Una vez entrenado el modelo, se guarda para poder ser utilizado en el fichero *valuate.py*.
- *valuate.py:* se realiza la clasificación (binaria o multiclase) del conjunto de test de la base de datos UNSW-NB15. Para cada instancia, se computa la función de pérdida para todas las clases posibles. La clase que arroje menor pérdida (menor distancia) es la elegida por el CVAE como clase predecida.

Una vez explicado de forma somera la función que realiza cada uno de los códigos, es pertinente entrar en profundidad en el desarrollo del modelo y las modificaciones realizadas. El primer código a ejecutar es *preprocessing.py*, encargado de cargar los archivos correspondientes al set reducido del UNSW-NB15 (UNSW_NB15_training-set.CSV y UNSW_NB15_testing-set.CSV), aplicar normalización Min-Max a las variables continuas para escalarlas al rango [0-1] y codificación *one-hot* a las variables categóricas. Se eliminan también las características *id*, *is_ftp_login*, *ct_ftp_cmd*, *trans_depth*, *is_sm_ips_ports* y *ct_flw_http_mthd* por ser en su mayoría 0. Además, como ya se explicó en el capítulo 4, antes de llevar a cabo la codificación *one-hot*, se verifica que cada uno de los conjuntos (entrenamiento y set) tiene los mismos elementos en las columnas *proto, service* y *state.* Si existe una entrada de *proto* en el set de entrenamiento que no se encuentra en el set de test, se añade a esta última como una columna vacía (y viceversa). Por último, antes de guardar ambos conjuntos de muestras ya procesados en formato .CSV, se realizan ciertos ajustes en los datos para tener conjuntos de datos separados dependiendo de si se quiere hacer clasificación binaria ("Normal", "Attack") o multiclase. Los conjuntos de datos de entrenamiento y set para clasificación binaria se llaman VAE_Train+.CSV y VAE_Test+.CSV respectivamente, mientras que para clasificación multiclase se llaman VAE_Train_M+.CSV y VAE_Test_M+.CSV.

A continuación, el código *model.py* es el que alberga la estructura de capas del CVAE y donde se define la reparametrización necesaria para poder aplicar *backpropagation* al modelo y poder actualizar los pesos. La estructura de capas del modelo debe ser cambiada ligeramente dependiendo de si la clasificación que se va a realizar es binaria o multiclase; una de las capas tendrá un número de entradas distinto en cada caso. La estructura de capas del codificador y decodificador es la que aparece en la Figura 51.

[20] *Véase Anexo B.*

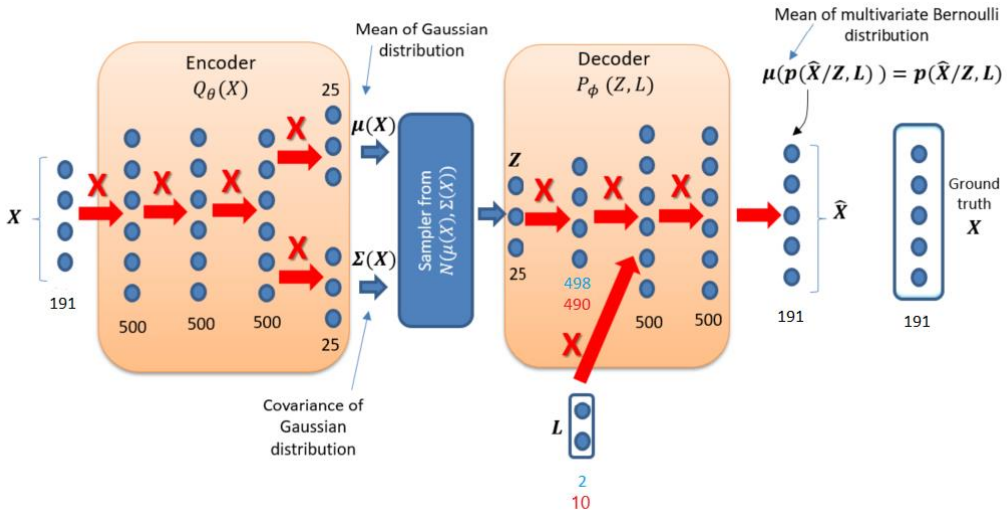


Figura 51: Estructura de capas del codificador y decodificador para el CVAE, adaptado a la UNSW-NB15. Los números en azul se corresponden con el tamaño de la entrada de la capa o longitud del vector de etiquetas de clase L para una clasificación binaria. Los números en rojo se corresponden con el tamaño de la entrada de la capa o longitud del vector de etiquetas de clase L para una clasificación multiclase. Figura adaptada de [100].

Las X rojas en el modelo indican que las capas están completamente conectadas entre sí, mientras que los números existentes debajo de cada capa designan el número de nodos de cada capa. La función de activación de todas las capas es *ReLu*, a excepción de la de la última capa del codificador (la cual es *Linear*) y la última capa del decodificador (que es *Sigmoid*). Entrando en detalles concretos del codificador, su primera capa tiene 191 nodos a la entrada puesto que éste es el número de características final del conjunto de datos (una vez se ha aplicado *one-hot* y se ha eliminado la columna de las etiquetas de clase). Los datos van pasando por una sucesión de capas completamente conectadas entre sí, hasta llegar a la capa de salida, donde se obtienen la media y la covarianza de $q_\phi(\boldsymbol{Z}|\boldsymbol{X})$, para la cual se emplea una distribución Gaussiana multivariante $N(\mu(\boldsymbol{X}),\Sigma(\boldsymbol{X}))$ [100]. Como distribución a priori para el espacio latente se utiliza una distribución normal $N(0,\boldsymbol{I})$.

Posteriormente, el decodificador muestrea de forma aleatoria en este espacio latente (aquí es donde entra el truco de la reparametrización explicado anteriormente, que permite tomar muestras del espacio latente, aplicar la propagación hacia atrás y poder actualizar los pesos de la red). Para la distribución $p_\theta(\mathbf{X}|\mathbf{Z},\mathbf{L})$ se utiliza una distribución Bernoulli multivariante. Esta distribución tiene la particularidad de no necesitar un muestreo final, ya que el parámetro de salida que caracteriza la distribución es su media, que es igual que su probabilidad de éxito p. Esta probabilidad de éxito p puede ser interpretada como un valor escalado entre [0-1] con respecto al conjunto de datos original $\boldsymbol{X}$, el cual ya se escaló entre [0-1] al preprocesarlo. En definitiva, la salida del decodificador se puede interpretar directamente como el conjunto de muestras reconstruidas por el modelo, $\hat{\boldsymbol{X}}$ [100].

Ahondando un poco más en la estructura del decodificador, la primera capa completamente conectada de éste tendrá un número distinto de nodos de salida dependiendo de si el entrenamiento o clasificación es binario o multiclase, ya que entre esta capa y la siguiente se insertan las etiquetas de clase (se ha determinado esta posición para insertar las etiquetas de clase de forma empírica, según [100]). En el caso binario, la capa tendrá 498 nodos de salida, a los que se les concatenará el vector L (que contiene las etiquetas de clase en codificación *one-hot).* Así encaja perfectamente con los 500 nodos que toma la siguiente capa del decodificador como entrada. En el caso multiclase, al contar con 10 etiquetas de clase en total (*normal, generic, exploits, fuzzers, DoS, reconnaissance, analysis, backdoor, shellcode, worms*), la primera capa del decoder tendrá 490 nodos de salida. Por tanto, hay que cambiar esta capa del modelo dependiendo de la opción escogida.

Por otro lado, el código *cvae.py* es el encargado de entrenar el modelo y prepararlo para la clasificación que se vaya a realizar en *valuate.py* (binaria o multiclase). Después de cargar los conjuntos de entrenamiento y test adecuados, y de invocar al modelo de CVAE definido en *model.py*, se realizan 100 epochs de entrenamiento. A medida que éste se realiza, se va computando también la función de pérdida en cada epoch (tanto para el conjunto de entrenamiento como para el de test). Concretamente, para cada epoch se realizan los siguientes pasos:

- Se llama a la función *train()*, que efectúa el entrenamiento del CVAE. Se divide en batches el conjunto de entrenamiento y sus respectivas etiquetas de clase (*x,y*), y se va pasando cada batch al modelo. El modelo devuelve las muestras del batch reconstruidas, así como la media y la desviación típica de $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Una vez llegados a este punto, se calcula la función de pérdida del modelo, la cual se busca minimizar:

$$Loss\ function = -\mathbb{E}\left[\log p_\theta\left(\widehat{\boldsymbol{X}}|\mathbf{Z},\mathbf{L}\right)\right] + KL\left(q_\phi(\mathbf{z}|\boldsymbol{x})|p_\theta(\mathbf{z})\right)$$

  Después de este paso, se computa el gradiente de la función de pérdida y se suma el valor de pérdida del batch al total de todos los batches. Por último, se llama al optimizador (en este caso, Adadelta, con una tasa de aprendizaje de 0.001) para actualizar los pesos del modelo. Una vez se aplica este procedimiento a todos los batches, *train()* devuelve la pérdida total dividida entre el número de muestras del conjunto de entrenamiento.

- Después, se llama a *test()*, donde se aplica el modelo entrenado en *train()*. Se divide en batches el conjunto de test y sus respectivas etiquetas de clase (*x,y*), y se va pasando cada batch al modelo. El modelo devuelve las muestras del batch reconstruidas, así como la media y la desviación típica de $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Una vez llegados a este punto, se calcula la función de pérdida del modelo y se suma el valor de pérdida del batch al total de todos los batches. En este caso, al estar con el conjunto de test, no se aplica cálculo de gradientes ni propagación hacia atrás. Por último, *test()* devuelve la pérdida total dividida entre el número de muestras del conjunto de test.

  Una vez se terminan los 100 epochs, se guarda el modelo entrenado y se grafican las pérdidas de entrenamiento y de test frente a los epochs, arrojando los siguientes resultados:

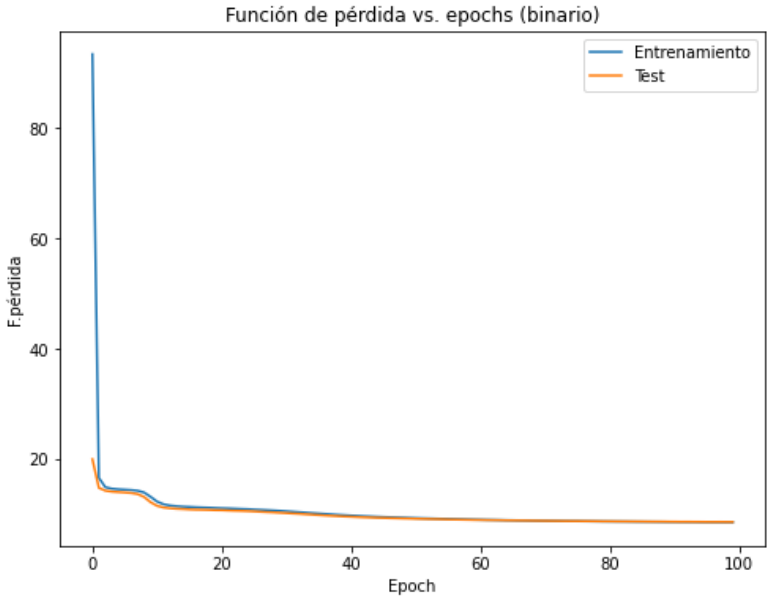


Figura 52: Gráfica que representa la función de pérdida frente a los epochs para un entrenamiento binario.

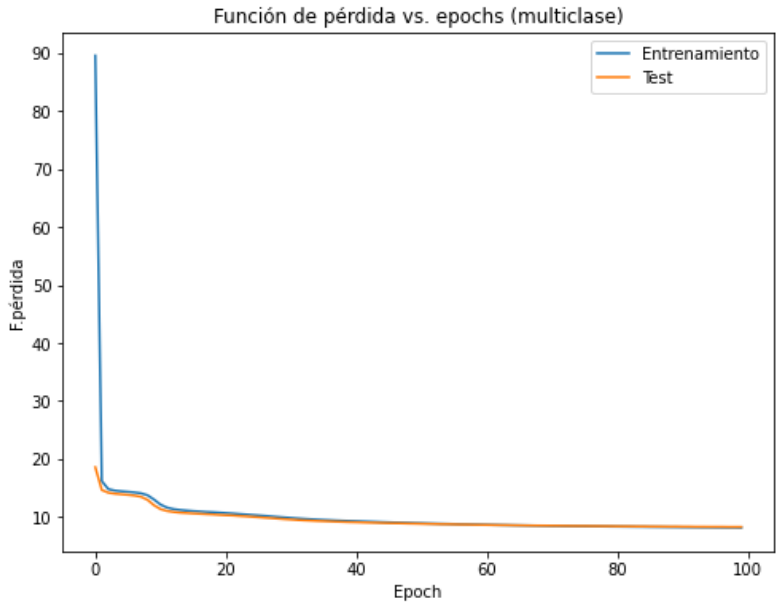


Figura 53: Gráfica que representa la función de pérdida frente a los epochs para un entrenamiento multiclase.

Después de 100 epochs, ambas gráficas convergen en torno al mismo valor de pérdida para el entrenamiento binario y multiclase (aproximadamente 8). Además, se puede observar cómo ambas figuras tienden a estabilizarse en torno a este valor, por lo que el aumento de epochs no conllevaría una mayor minimización de la función de pérdida.

En último lugar, el código *valuate.py* es el que realiza la clasificación binaria o multiclase del conjunto de test correspondiente (VAE_Test+.CSV en el caso de clasificación binaria y VAE_Test_M+.CSV en el caso de la clasificación multiclase). Después de cargar el conjunto de datos correspondiente, separar las instancias y las etiquetas de clase y cargar al modelo de CVAE los parámetros (pesos y sesgos) del modelo entrenado en *cvae.py*, se clasifica cada instancia del conjunto de test tal y como se detalló en el apartado 5.4.2 de este capítulo. Por último, se llama al código *metricas.py* para obtener los resultados de clasificación binaria (o multiclase) del CVAE.

Los resultados obtenidos para una clasificación binaria (métricas y matriz de confusión) son los siguientes:

```
El tiempo de ejecución fue: 113.422 segundos.
Estadísticas por categoría y generales:

              precision    recall  f1-score   support

           0     0.6056    0.6250    0.6152     37000
           1     0.6857    0.6678    0.6766     45332

    accuracy                         0.6486     82332
   macro avg     0.6457    0.6464    0.6459     82332
weighted avg     0.6497    0.6486    0.6490     82332
```

Figura 54: Resultados obtenidos por el CVAE para una clasificación binaria. "El valor "0" se corresponde con tráfico normal, mientras que el valor "1" se corresponde con tráfico anómalo (ataques). El campo "support" muestra el número de instancias existentes de cada clase.

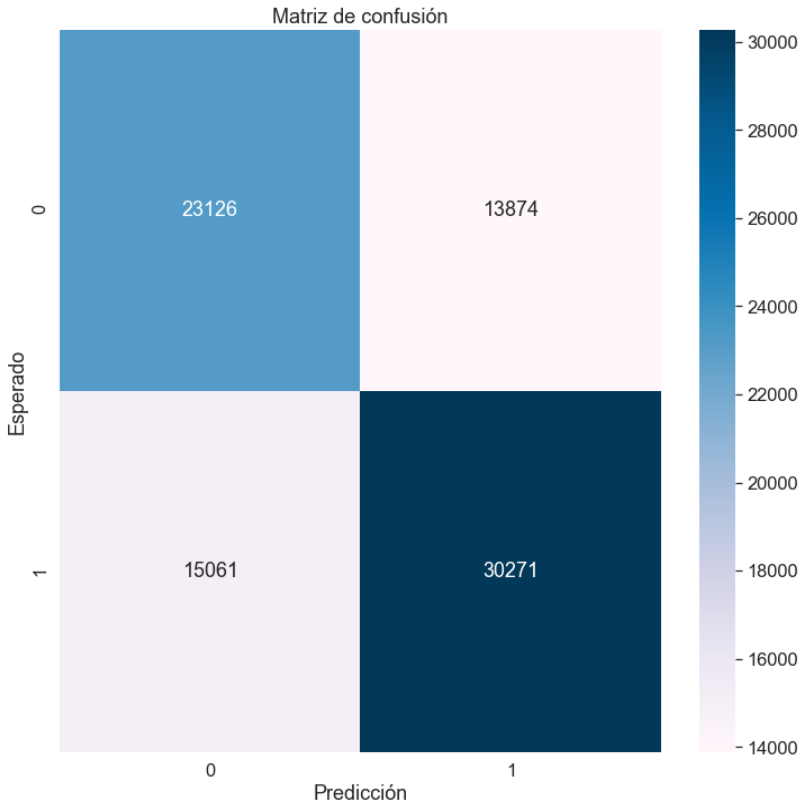


Figura 55: Matriz de confusión correspondiente a los resultados del CVAE como clasificador binario.

Los resultados obtenidos para una clasificación multiclase (métricas y matriz de confusión) son los siguientes:

```
El tiempo de ejecución fue: 513.568 segundos.
Estadísticas por categoría y generales:

              precision    recall  f1-score   support

      Normal     0.9664    0.1969    0.3272     37000
     Generic     0.3046    0.1733    0.2209     18871
    Exploits     0.2216    0.1419    0.1730     11132
    Fuzzers      0.1431    0.0582    0.0828      6062
         DoS     0.0555    0.0651    0.0599      4089
Reconnaissance   0.0577    0.0870    0.0694      3496
    Analysis     0.0093    0.1610    0.0175       677
    Backdoor     0.0080    0.1492    0.0152       583
   Shellcode     0.0062    0.1111    0.0118       378
       Worms     0.0003    0.1136    0.0007        44

    accuracy                         0.1616     82332
   macro avg     0.1773    0.1257    0.0978     82332
weighted avg     0.5500    0.1616    0.2334     82332
```

Figura 56: Resultados obtenidos por el CVAE para una clasificación multiclase. "support" indica el número de muestras del conjunto de test pertenecientes a cada clase.

Figura 57: Matriz de confusión correspondiente a los resultados del CVAE como clasificador multiclase.

De los resultados se extrae en primer lugar que el CVAE no consigue clasificar correctamente un gran número de muestras para el caso binario, obteniéndose unos resultados de en torno al 65% de F1-Score. El resto de métricas extraídas obtienen prácticamente el mismo valor. Los resultados empeoran mucho más cuando la clasificación es multiclase: el clasificador incurre en numerosos fallos a la hora de clasificar muestras de forma generalizada. Dentro de estos malos resultados en general, se aprecia la misma tónica que en la clasificación multiclase realizada en el Artículo II: en general, clasifica peor las clases minoritarias que las mayoritarias. Esto se puede apreciar si se observa el F1-Score para cada una de las clases del conjunto de datos. A la vista de los números obtenidos, el modelo del autocodificador variacional condicional desarrollado no ofrece un buen rendimiento como clasificador binario ni multiclase, sobre todo en comparación con los dos artículos del algoritmo de *decision tree* desarrollados en el capítulo anterior (sus métricas estaban cercanas al 90% en tareas de clasificación binaria y al 40% en clasificación multiclase). Quizá habría que intentar modificar la estructura de capas del codificador y decodificador añadiendo más capas ocultas para procesar los datos, o intentar equilibrar el número de muestras en cada clase con un generador sintético para ver si así el desempeño del CVAE mejora.

Finalmente, para terminar de comparar el rendimiento del CVAE como clasificador con respecto a otros modelos, en el apartado siguiente se lleva a cabo la construcción de un perceptrón multicapa como clasificador binario y multiclase. Como cierre, se establecerá una comparación global entre los resultados obtenidos por el CVAE y el resto de modelos desarrollados.

5.4 Desarrollo y evaluación de un MLP

Además de la comparación de los resultados del CVAE con los modelos de *random forest*, se propone en este apartado la construcción de un algoritmo clasificador más cercano en lo que a estructura se refiere al autocodificador variacional condicional: un perceptrón multicapa. Como referencia para realizar el perceptrón multicapa, se usarán algunos de los parámetros utilizados en el artículo "*Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset*", de los autores Sydney Kasongo y Yanxia Sun [101]. En el punto 5.2.1 de este capítulo ya se explicó el funcionamiento y configuración de una red neuronal profunda con detalle (básicamente MLP y DNN hacen referencia al mismo concepto).

Recapitulando brevemente, se puede decir que un perceptrón multicapa se compone de una capa de entrada, una o más capas ocultas y una capa de salida. Todas las capas, a excepción de la de salida, tienen un sesgo y están completamente conectadas a la siguiente capa. Cada muestra del conjunto de instancias alimenta a la red y se calcula la salida de cada neurona en cada capa consecutivamente. Al llegar a la capa de salida, se mide el error de salida de la red (la diferencia existente entre la salida deseada y la salida real de la red) y se computa la contribución de cada neurona de la última capa oculta al error de cada neurona de la capa de salida. Después, se mide la contribución de las neuronas de la capa oculta anterior al error de salida, y así sucesivamente hasta llegar a la capa de entrada (este es el proceso conocido como propagación hacia atrás o *backpropagation*). Así, se mide el gradiente de error a través de todos los pesos de la red, propagando el gradiente de error hacia atrás y reajustando los pesos de la red para reducir el error. [40]

El código desarrollado es *mlp.py*[21], cuya estructura es idéntica al código *articulo_II.py* desarrollado en el Capítulo 4 para el *random forest* (a excepción del clasificador). Para evaluar las prestaciones del MLP, se utiliza el set reducido de la UNSW-NB15 (UNSW_NB15_training-set.CSV y UNSW_NB15_testing-set.CSV). En primera instancia, se realizaron las simulaciones atendiendo a los parámetros que, según [101], mejores resultados arrojaban: selección de las 19 mejores características consideradas por los autores, una sola capa oculta de 150 neuronas, tasa de aprendizaje adaptativa, y de valor 0.02, y Adam como optimizador escogido. Sin embargo, los resultados obtenidos con esta configuración inicial no fueron buenos. Con objetivo de intentar mejorarlos, se optó entonces por cambiar la configuración original, aumentando el número de capas ocultas, así como el número de neuronas; además, se disminuye la tasa de aprendizaje a 0.002. Con estos cambios se consiguió aumentar (en torno a un 5-7%) el F1-Score general. Finalmente, la configuración con la que se han conseguido los mejores resultados posibles es la que se muestra en la Figura 58.

```
mlp_clf = (MLPClassifier(hidden_layer_sizes=(600,400,200,100,50),
                         learning_rate_init=0.002,activation=act,
                         learning_rate="adaptive",solver='adam',verbose=1))
```

Figura 58: Mejor configuración de hiperparámetros del MLP para clasificación binaria y multiclase. La función de activación óptima (parámetro act) es una sigmoide en clasificación binaria, y ReLu en clasificación multiclase).

Los números ofrecidos por el MLP como clasificador binario siguen más o menos la misma tendencia de resultados que el autocodificador variacional condicional. Como ocurría con el CVAE, el modelo construido no obtiene unos resultados muy efectivos en la clasificación de instancias en las clases normal y ataque; no obstante (y a diferencia del CVAE) se muestra mucho más acertado a la hora de discernir de forma correcta las instancias pertenecientes a ataques (se obtiene casi un 82% de F1-Score).

[21] *Véase Anexo B.*

A pesar de clasificar correctamente la mayoría de instancias pertenecientes al tráfico anómalo, la matriz de confusión mostrada en la Figura 60 permite visualizar de forma clara los problemas de clasificación que el modelo tiene de forma general con las muestras del conjunto de datos, especialmente con el tráfico normal: aproximadamente un tercio de las muestras correspondientes a esta categoría han sido clasificadas erróneamente como tráfico anómalo.

En general, los valores de exactitud, precisión, sensibilidad y F1-Score están cercanos a un 80%. Estos números quedan por debajo de los obtenidos en los dos modelos de *decision tree*, pero sí que están por encima de los números conseguidos por el CVAE. En cuanto a tiempo de ejecución, el modelo tarda en efectuar el entrenamiento del modelo y predicción de clases en el caso binario casi una hora.

```
El tiempo de ejecución fue: 3285.192 segundos.
Estadísticas por categoría y generales:

              precision    recall  f1-score   support

           0     0.8159    0.6718    0.7369     37000
           1     0.7659    0.8763    0.8174     45332

    accuracy                         0.7844     82332
   macro avg     0.7909    0.7741    0.7771     82332
weighted avg     0.7884    0.7844    0.7812     82332
```

Figura 59: Resultados obtenidos por el MLP para una clasificación binaria. El valor "0" se corresponde con tráfico normal, mientras que el valor "1" se corresponde con tráfico anómalo (ataques). El campo "support" muestra el número de instancias existentes de cada clase.

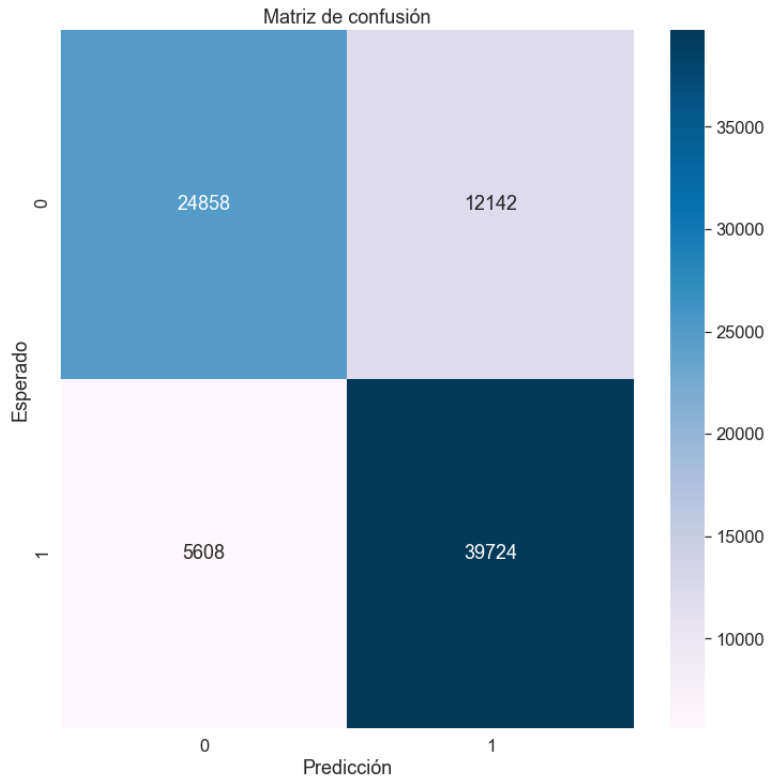


Figura 60: Matriz de confusión correspondiente a los resultados del MLP como clasificador binario.

Por otro lado, los resultados arrojados por el MLP como clasificador multiclase se muestran en las siguientes figuras:

El tiempo de ejecución fue: 8925.871 segundos.
Estadísticas por categoría y generales:

	precision	recall	f1-score	support
Normal	0.9263	0.3344	0.4914	37000
Generic	0.6671	0.5834	0.6225	18871
Exploits	0.2213	0.5702	0.3188	11132
Fuzzers	0.1955	0.5262	0.2851	6062
DoS	0.0169	0.0201	0.0183	4089
Reconnaissance	0.0061	0.0003	0.0005	3496
Analysis	0.0377	0.0207	0.0267	677
Backdoor	0.0833	0.0069	0.0127	583
Shellcode	0.0280	0.1508	0.0473	378
Worms	0.0000	0.0000	0.0000	44
accuracy			0.4018	82332
macro avg	0.2182	0.2213	0.1823	82332
weighted avg	0.6156	0.4018	0.4291	82332

Figura 61: Resultados obtenidos por el MLP para una clasificación multiclase. "support" indica el número de muestras del conjunto de test pertenecientes a cada clase.

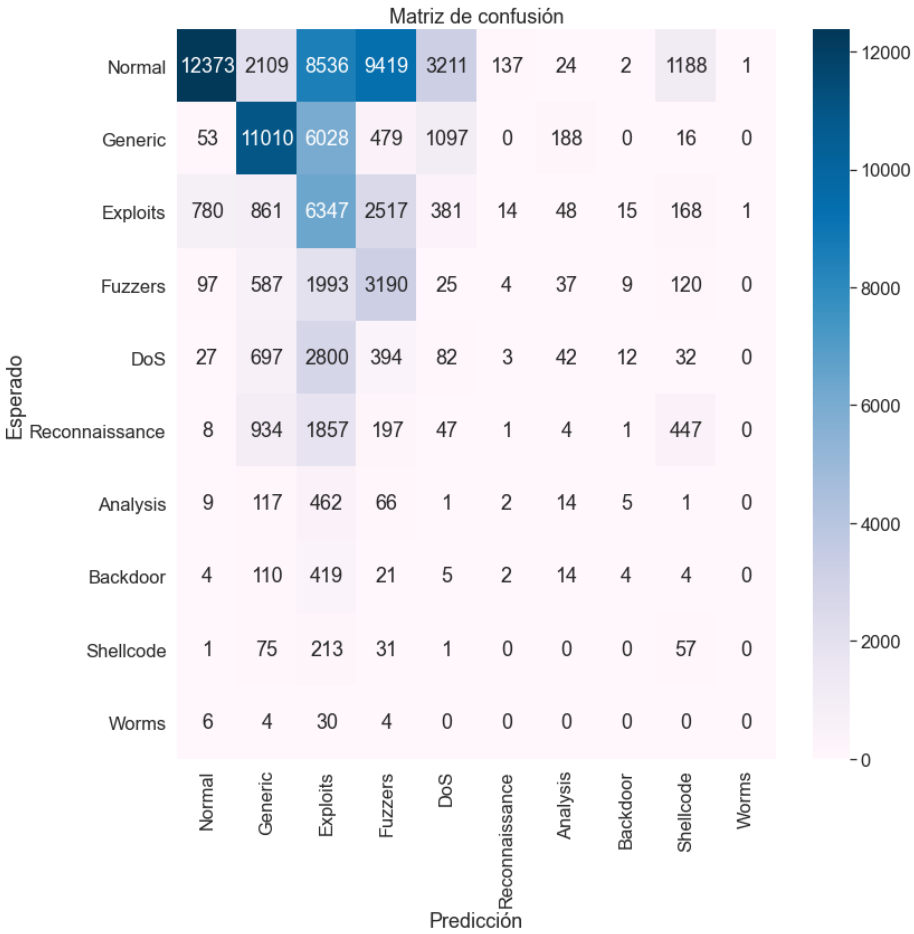


Figura 62: Matriz de confusión correspondiente a los resultados del MLP como clasificador multiclase.

Los números ofrecidos por el MLP como clasificador multiclase siguen la tónica general que se ha visualizado en todas las clasificaciones de este tipo, desempeñándose mejor en las clases más numerosas y cometiendo muchos fallos en las clases minoritarias, generalmente. Resulta llamativo el hecho de la cantidad de falsos positivos registrados en muchas clases: el modelo no es capaz de clasificar correctamente casi 25.000 instancias de la clase *Normal,* etiquetándolas como *fuzzers,* y *exploits,* y en menor medida, *generic, DoS* y *shellcode.* El CVAE también experimentaba este fallo, pero, mientras que el MLP clasifica de forma errónea las instancias de *normal* en 3 categorías mayormente, el CVAE repartía las muestras mal clasificadas de *normal* entre prácticamente todas las clases. Además, comparando ambas matrices de confusión (Figura 62 y Figura 57) se visualiza claramente que el CVAE no sólo hace esto con la clase *normal,* si no también con otras clases con gran número de instancias como *generic* y *exploits.*

Dentro de los malos resultados, este modelo vuelve a estar ligeramente por encima en tareas clasificatorias en comparación con el autocodificador variacional condicional (sus métricas están un 10% aproximadamente por encima de las del CVAE).

En cuanto a tiempo de ejecución, el modelo tarda en efectuar el entrenamiento del modelo y predicción de clases en el caso multiclase en torno a 2 horas.

5.5 Comparación de resultados de CVAE con el resto de modelos y conclusiones

Una vez mostrado el desempeño del autocodificador variacional condicional y de los otros dos modelos desarrollados (*decision tree* y MLP) en ambos tipos de clasificación, es hora de enfrentar entre sí todos los resultados obtenidos para visualizar de forma global las conclusiones que se pueden extraer referentes al rendimiento del CVAE como clasificador de ataques a una red. Es necesario recordar que los conjuntos de datos de los dos artículos de *random forest* son distintos a los usados por el CVAE y MLP para la clasificación, dato importante para comprender la comparativa teniendo este detalle en consideración. Se ha escogido para establecer la comparación la media *macro* en las métricas, ya que, debido a que la base de datos está fuertemente imbalanceada, iba a ponderar con mayor peso las métricas con mayor número de instancias, que son principalmente las que mejor clasifican los modelos. Este ponderado (como ya se explicó en su momento) toma de forma equitativa todas las clases.

De la observación de las Figuras 63 y 64, que engloban los resultados obtenidos por cada uno de los modelos en clasificación binaria y multiclase, se puede concluir que, con diferencia, el mejor modelo de los 3 desarrollados en este trabajo es el *decision tree*. Ambos artículos, el I (que se entrenaba y testaba con el UNSW-NB15-training-set.csv) y el II (que se entrenaba y testaba con el conjunto de datos completo del UNSW-NB15, más de 2 millones y medio de muestras), son los que mejores métricas arrojan, tanto en clasificación binaria como multiclase.

Particularizando para cada caso, en cuanto a clasificación binaria el CVAE (aunque relativamente cerca del MLP) queda por debajo de todos los modelos en cuanto a resultados, con las cuatro métricas extraídas (exactitud, precisión, sensibilidad y F1-Score) en torno al 65%. En el otro lado, los modelos de *decision tree* desarrollados están cercanos o por encima del 90% en las métricas, distinguiendo bastante bien entre tráfico normal y anómalo. Para la clasificación multiclase vuelve a observarse el mismo comportamiento, aunque con peores números de forma general. Todos los modelos desarrollados (*decision tree*, CVAE y MLP) han sufrido en mayor o menor medida a la hora de predecir ciertas clases, sobre todo las minoritarias. Poniendo el foco en la F1-Score (que es un compendio de la sensibilidad y la precisión), el modelo de árboles de decisión consigue un 42% de F1-Score, mientras que para los dos algoritmos de aprendizaje profundo el valor de la métrica desciende hasta la mitad, no

llegando ni a un 10% en el caso del CVAE. En definitiva, para ambas clasificaciones el CVAE no obtiene buenos resultados para la base de datos UNSW-NB15.

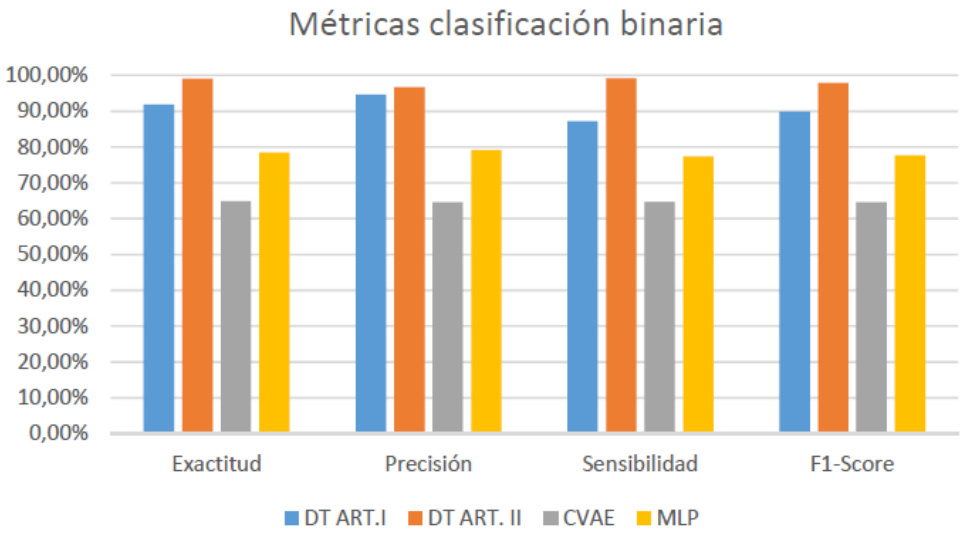


Figura 63: Comparativa global de resultados entre todos los modelos para una clasificación binaria.

Siguiendo con los comentarios acerca de los resultados en clasificación multiclase, esta vez desgranados por clases, todos los modelos han tenido problemas en mayor o menor medida al predecir las clases con menor número de instancias en el conjunto de datos. Dentro de estos malos resultados, el algoritmo que mejor vuelve a predecir las 10 clases es el *decision tree*, sobre todo las clases *normal* y *generic*. Estas dos clases también son las que mejor predicen tanto el MLP como el CVAE. Por el contrario, las clases *backdoor* y *analysis* causan dificultades a los tres modelos.

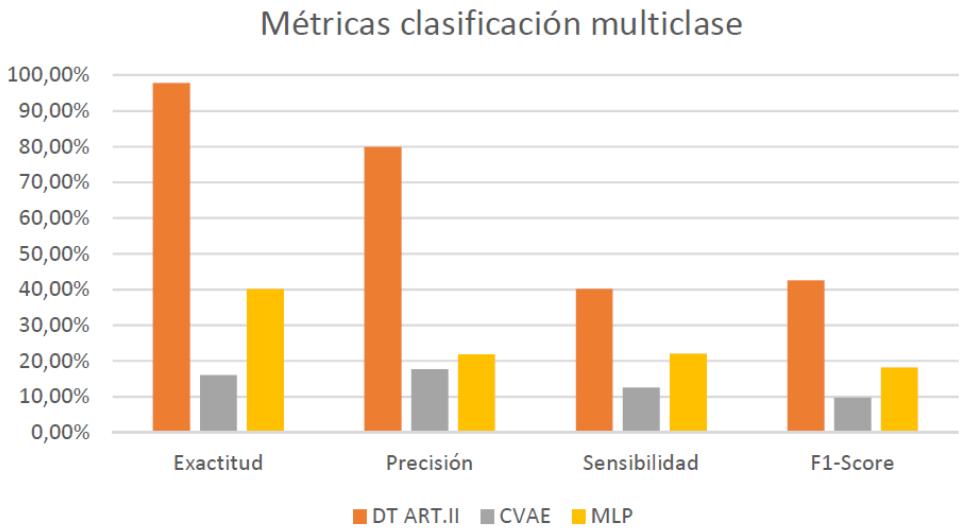


Figura 64: Comparativa global de resultados entre todos los modelos para una clasificación multiclase.

En vista del problema que ocasiona que las clases no estén balanceadas entre sí (sobre todo en el caso multiclase), se intuye que si se aplica un algoritmo generador de muestras sintéticas para equilibrar el número de instancias existentes en cada clase el rendimiento del CVAE subirá, y será capaz de discernir mejor entre clases. La conclusión final que se puede deducir a la vista de estos resultados es que hay otros algoritmos que resultan más eficaces a la hora de clasificar ataques a una red de comunicaciones (por lo menos, en lo que a la base de datos UNSW-NB15 respecta).

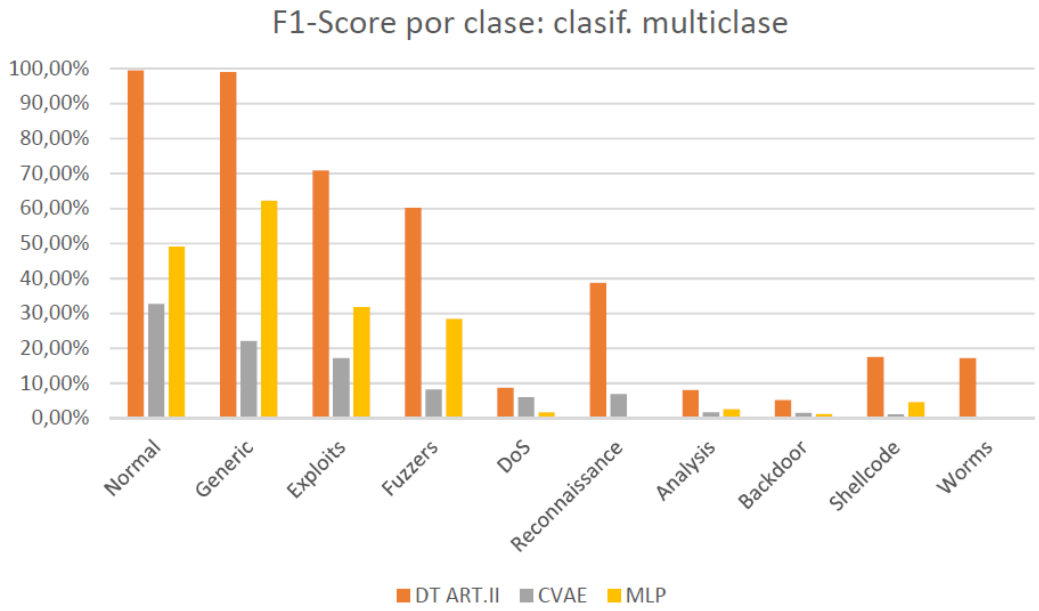


Figura 65: Comparativa global de resultados según clases tomando como referencia el F1-Score.

6 EXPLOTACIÓN DE UN IDS EN LA REALIDAD

Una vez presentados los resultados prácticos obtenidos en este trabajo, en este capítulo se aporta un enfoque que va más allá del punto de vista meramente académico. El último fin de un clasificador como los que se han presentado a lo largo de los capítulos del presente documento es formar parte de un sistema de detección de intrusiones plenamente operativo, que pueda ser usado por cualquier empresa como mejora de la seguridad de su infraestructura de red. Dentro de un sistema de detección de intrusos, el software que monitoriza el tráfico y detecta la actividad maliciosa (los clasificadores desarrollados en apartados anteriores formarían parte de ese software) probablemente sea la pieza central en torno a la que gira toda la infraestructura, pero no es la única. Como en el desarrollo de cualquier proyecto, se necesita conocer con todo lujo de detalles el entorno en el que se desplegará el sistema, los objetivos y requisitos exigidos por el cliente para poder, en última instancia, seleccionar el sistema concreto que éste necesita, selección de los componentes adecuados para implantar la solución, implementación y prueba del sistema al completo, etc. En definitiva, este capítulo pretende explicar cómo se llevaría a cabo la explotación de los clasificadores que se han estudiado o, en otras palabras, aportar ciertas líneas generales sobre los pasos a seguir para la implementación real de un sistema de detección de intrusos en su totalidad.

6.1 Análisis de requisitos y diseño

En primer lugar y antes de llevar a cabo la implementación física de un sistema de detección de intrusos, es necesario un estudio pormenorizado de las necesidades del cliente. Este paso resulta de vital importancia para el desarrollo del proyecto, y el objetivo principal es llegar a la solución que mejor se adapte a las necesidades que tiene la empresa que va a contratar el servicio. Para ello, es necesario saber qué tipo de IDS quiere instalar el cliente, el tipo de detección que se quiere emplear[22] o el mecanismo de respuesta una vez se detecta el ataque [102]. El objetivo de la detección de los ataques es otro factor a tener en cuenta a la hora de diseñar la solución final, puesto que si se desean emprender acciones legales contra los atacantes que intenten boicotear la red, es indispensable generar un formato adecuado de los registros que genera el sistema, así como asegurar su conservación. Si solamente se desea dotar de seguridad al sistema sin tomar represalias legales, no es necesario guardar datos de auditorías.

En los apartados que siguen a continuación se profundizará algo más acerca del modelo de funcionamiento de un sistema de detección de intrusos y los pasos que hay que seguir para la construcción de un software dedicado a solventar los problemas de seguridad de un cliente. Además, se proporcionará una descripción del resto de elementos que componen un sistema de este tipo.

[22] *Ver Capítulo 1.*

6.1.1 Modelo de funcionamiento de un sistema de detección de intrusos[23]

En la actualidad existen muchos softwares comerciales dedicados a la detección de ataques en una red (en apartados posteriores se mencionarán algunos de ellos). No obstante, si se quisiese realizar desde cero un programa dedicado a esta tarea, es conveniente seguir un modelo para desarrollarlo. El modelo más aceptado para la construcción de un sistema de detección de intrusos tiene tres fases principales: la fase de recogida de información, la fase de análisis y la fase de respuesta. En otras palabras, para que un sistema de detección de intrusos realice sus funciones de forma correcta debe realizar en primer lugar una recopilación de información, que posteriormente será procesada y analizada a través de diversas técnicas y, en última instancia, dará una respuesta a las amenazas detectadas. La figura 50 representa el esquema general de un modelo de gestión de seguridad, donde la detección de intrusiones estaría representada en el segundo paso de dicho esquema [102]:

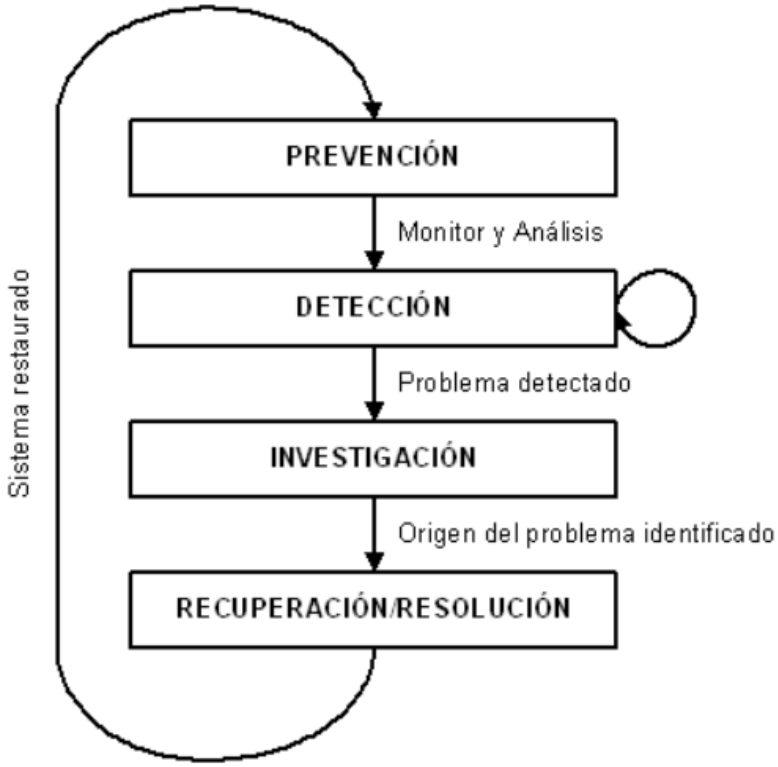


Figura 66: Modelo general de gestión de seguridad. Fuente: [102]

I. Recolección de datos

La primera fase para el desarrollo de un sistema de detección de intrusos consiste en recopilar información acerca del entorno donde se piensa implementar el IDS. Dependiendo del tipo de IDS que se adopte como solución, las fuentes de información son variadas, aunque los dos tipos predominantes son [102]:

[23] *Se suele denominar sistema de detección de intrusos (o IDS) tanto al sistema completo en sí como al software específico que realiza la clasificación del tráfico de red.*

- Basadas en máquina o host-based: Este tipo de fuentes de información consisten esencialmente en registros generados por las distintas aplicaciones del sistema operativo y los registros de sistema.
- Basadas en red: Probablemente sea la fuente de información más usada en este ámbito. En este caso, se captura el tráfico de red a través de un *sniffer* o rastreador (esto ayuda a no mermar el rendimiento de los equipos de la red). Este tipo de fuente de información aporta datos relevantes sobre ataques que no podrían ser detectados por un IDS basado en máquina (como, por ejemplo, ataques DoS).

II. Análisis

Una vez se han obtenido los datos de las distintas fuentes de información, se evalúan con distintas técnicas. Esencialmente, estas técnicas ordenan la información recabada, la analizan de forma estadística y la comparan con patrones de actividad sospechosa. En las siguientes líneas se detallan los objetivos y requisitos para llevar a cabo esta fase, así como los factores a tener en cuenta a la hora de detectar los ataques y los pasos necesarios para construir un modelo para analizar la información recogida (es decir, la elaboración de un software IDS).

▪ Objetivos principales

El objetivo principal de todo sistema de detección de intrusos es participar en la mejora de la seguridad de una red. Las metas esenciales que se deben cumplir en el desarrollo de esta fase de análisis son [102]:

- *Establecimiento de comportamientos:* Una de las funciones esenciales de esta fase es el estudio del comportamiento de la información recogida de forma estadística. Gracias a este estudio, se puede discernir entre lo que se considera como tráfico normal y anormal para poder detectar así posibles ataques o intrusiones en el sistema.

- *Control de calidad:* Mediante el análisis se pueden detectar problemas existentes en el diseño de la red o en la gestión que se está realizando de ella, y así corregirlos.

- *Información necesaria en intrusiones:* A veces es necesario que la información esté lo suficientemente detallada y bien ordenada para así poder emprender acciones legales en contra de los ciberatacantes.

▪ Requisitos y objetivos secundarios

Esta fase requiere del cumplimiento de al menos dos requisitos para que funcione correctamente. Una de ellas es la responsabilidad, es decir, la capacidad de asociar una actividad con la persona u objeto responsable de ella. Para conseguirlo se necesita un sistema de autentificación e identificación en el sistema cuyo funcionamiento no resulta nada fácil a nivel de red, ya que las acciones de un mismo usuario pueden tener origen en múltiples equipos.

El otro factor que debe cumplirse es la detección en tiempo real y respuesta. Se debe detectar el tráfico que corresponde a actividad maliciosa y reaccionar ante éste, bloqueándolo o restaurando el sistema al estado previo antes del ataque.

Esta fase también puede tener otros objetivos diferentes a los mencionados en el apartado anterior: por ejemplo, la información recabada puede ser usada para la elaboración de análisis del sistema, o bien para monitorizarlo y optimizar su funcionamiento [102].

- **Factores que intervienen en la detección de ataques**
  Para realizar el análisis de datos resulta útil comprender los distintos elementos que pueden intervenir en la detección de intrusiones [102]:

  - *Factor humano:* El ser humano ha quedado relegado a un segundo plano a la hora de intervenir en la detección de ataques debido a la irrupción de los IDS. Diversos estudios han demostrado que durante un período de tiempo determinado en el que un IDS es capaz de detectar miles de ataques, un humano solamente detecta alrededor del 2% de las mismas.

  - *Eventos externos:* Cualquier evento externo al sistema puede ser relevante a la hora de analizar el sistema en busca de eventos sospechosos como, por ejemplo, la detección de un crecimiento inusual en los informes de anomalías de un determinado sistema.

  - *Preámbulos de intrusiones:* Se puede averiguar si un sistema va a ser atacado si presenta ciertas evidencias de intrusiones.

  - *Artefactos de intrusiones:* Algunos ejemplos de evidencias de intrusiones pueden ser fallos del sistema sin motivo alguno, o la aparición de ficheros corruptos.

  - *Tiempo:* El hecho de poder monitorear el sistema en tiempo real hizo que la detección de ataques mejorase de forma bastante importante.

- **Desarrollo de un modelo para la detección de ataques**
  Una vez se tienen claros los objetivos, requisitos y demás factores a tener en cuenta para la creación de un IDS, se comienza con el desarrollo del algoritmo de detección de ataques que conformará el núcleo del software IDS. Concretamente, este apartado es el que se ha venido desarrollando a lo largo de todo este documento con los algoritmos de *random forest* y CVAE.

  El modelo propuesto por Rebecca Bace es uno de los más comunes y se estructura en 3 pasos [102]:

  - *Construcción del analizador:* En esta fase se construye el analizador o clasificador para la detección de ataques. En primer lugar, se recopilan y generan eventos, los cuales posteriormente se transforman y preprocesan a un formato único. Estos eventos recogidos pueden haberse generado en condiciones controladas, como en un laboratorio, o incluso hechos a mano.
    Una vez hecho esto se crea el clasificador que discriminará dichos eventos, discerniendo entre el tráfico normal y el tráfico sospechoso. Después de construirlo, se le pasan los datos previamente preprocesados para entrenarlo. Por último, los datos generados por el analizador conformarán la base de datos de conocimiento, sobre la cual se comparará el tráfico real analizado para determinar si el tráfico es normal o no.

  - *Realización del análisis:* En este paso se inserta el clasificador en el sistema a monitorizar, comenzando a trabajar con el tráfico en tiempo real que se intercambia a través de la red. El flujo de datos se preprocesa y se contrasta con la base de datos de conocimiento para determinar si los eventos detectados son normales o son ataques. En caso de detectar una intrusión, se genera una respuesta (una alarma, una acción de respuesta al ataque…)

- *Refinamiento del proceso:* Este último paso se ejecuta en paralelo a la fase de realización del análisis. Mientras el analizador está funcionando, se pueden optimizar los patrones de detección de ataques, o determinar otro tipo de parámetros a configurar.

III. Respuesta

Los resultados arrojados por el analizador se usan para ejecutar acciones. El tipo de respuestas a considerar dependen de muchos factores; por ejemplo, el tipo de entorno en el que se encuentra instalado el IDS. No tiene las mismas necesidades de notificaciones y alarmas un IDS que monitorice el tráfico de múltiples hosts de una red que el instalado en un ordenador personal. También se deben tener en cuenta ciertos procedimientos que estén instaurados en la red: en entornos militares, si el IDS no está activo en una red, se indica que esa red no puede trabajar con información confidencial.

Las respuestas que se pueden proporcionar a los ataques son activas o pasivas. Las respuestas activas pueden ser ejecutadas por el propio IDS o por intervención humana, y se pueden clasificar en 3 grupos principalmente: ejecución de acciones contra el intruso, restauración del sistema y recopilación adicional de información para emprender acciones legales. Por otra parte, las respuestas pasivas consisten en el envío de toda la información acerca del ataque a la persona encargada del sistema, dejando que ésta decida sobre cómo actuar [102].

6.1.2 Elección de elementos del sistema completo

El analizador de ataques o software IDS no es el único elemento a tener en cuenta a la hora de implementar un sistema de detección de intrusos.

En la figura 51 se puede observar un sencillo diagrama que representaría la estructura de un sistema de detección de intrusos basado en red (el más habitual) y los dispositivos físicos que la componen, que serían dos: la máquina, que sería el equipo final donde el responsable de seguridad vería las alertas generadas por el software IDS, y el dispositivo sonda, que suele tener instalados tanto el agente que se encarga de capturar el tráfico de red como el software IDS. Pero no solo es necesario escoger el hardware adecuado para ambos dispositivos; también se deben de tomar decisiones sobre qué software usar en la solución diseñada. La base de datos donde se guardan las alertas, el software IDS que se usará, la interfaz gráfica y el recolector de alertas también deben de escogerse de acuerdo a la configuración que se desea implementar.

Para visualizar dónde encajan en el sistema todas estas aplicaciones, resulta conveniente comentar la ruta que sigue el tráfico de red desde que entra en el sistema monitorizado hasta que el responsable de seguridad de la empresa ve las alertas generadas y toma acciones en consecuencia:

- En primer lugar, los paquetes son escaneados por el agente del IDS. Uno de los objetivos del agente es no ser reconocido en la red por los ciberatacantes; por este motivo la interfaz de red dedicada a la escucha se configura de forma que no tenga IP.
- Una vez ha recogido el tráfico, éste se pasa al IDS, el cual lo procesa, clasifica y genera alertas.
- Estas alertas son recogidas por un recolector de alertas y se envían a la base de datos que se encuentra en la máquina. En último lugar, la interfaz gráfica genera un entorno agradable para que el responsable de seguridad de la empresa visualice las alertas enviadas por el IDS y decida sobre cómo actuar en consecuencia.

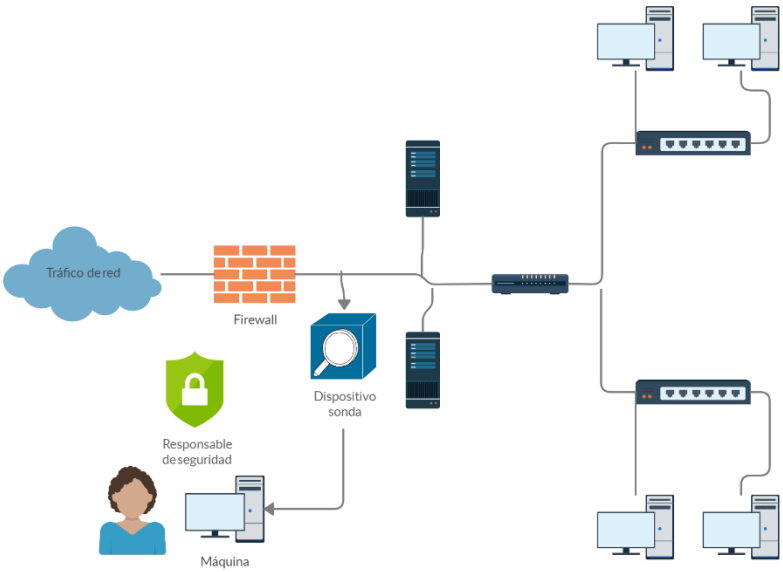


Figura 67: Diagrama de alto nivel del camino que sigue el flujo de tráfico cuando existe un IDS en la red.

I. IDS comerciales y opensource existentes en la actualidad

En los apartados anteriores se han especificado las ideas generales del modelo a seguir para la construcción de un software IDS desde cero, aunque, por otra parte, a día de hoy existe una gran variedad de IDS comerciales y de código abierto (existen otros IDS de elevadas prestaciones y de pago como el de McAfee). Los modelos desarrollados en el presente trabajo son un prototipo de los IDS que se presentan a continuación. De entre todos ellos destacan los siguientes:

- **Snort**

  Es una de las opciones más asentadas y extendidas de motores IDS. Según SC Magazine, su gran éxito se debe a la gran comunidad sobre la que se sustenta su código abierto, que detectan y dan respuesta de forma rápida y eficiente a los problemas de seguridad planteados, en comparación con otros IDS. Millones de descargas corroboran el triunfo de este software dedicado a la detección de intrusiones.

  El principal trabajo de Snort es buscar firmas en el tráfico de red que se ha captado que puedan indicar un problema de seguridad en el sistema. Al usar un lenguaje basado en reglas, éstas están configuradas para dar respuesta activa o pasiva. Las empresas pueden aprovechar el conjunto de reglas proporcionadas por el software de forma predeterminada o bien, escribir o modificar sus propias reglas para adaptarlas a las necesidades de su red. Gracias a la gran comunidad existente detrás de esta aplicación, las reglas están mejorándose y revisándose día tras día para estar al tanto de los últimos ciberataques conocidos [103].

- **Bro IDS**

  Este software fue desarrollado por el estadounidense Vern Paxson del Laboratorio Nacional de Berkeley junto con el Instituto Internacional de Ciencias Informáticas. Bastante similar a Snort, Bro detecta intentos de intrusión mediante la búsqueda de patrones sospechosos en el tráfico de red. Para conseguir detectar los ataques en tiempo real, este software utiliza dos interfaces de red (una en dirección a internet y otra en dirección a la red interna) para capturar todo el tráfico. Con el hardware adecuado y la configuración correcta del sistema operativo donde está instalado, Bro es capaz de realizar detección en tiempo real sin obstaculizar la velocidad de la red [104].

- **Suricata**

  Este software contó con el apoyo financiero del Departamento de Seguridad de Estados Unidos, que quería crear una alternativa multiproceso a Snort con el objetivo de proteger las redes contra intrusiones más avanzadas. La arquitectura "multithread" de Suricata hace que pueda soportar sistemas multi-core y multiprocesador de alto rendimiento. Al disponer de múltiples hilos, ofrece mucha más velocidad y eficiencia a la hora de efectuar el análisis del tráfico de red, y puede ayudar a dividir la carga de trabajo dependiendo de las necesidades de procesamiento del sistema. Unido a esto, el software está diseñado para aprovechar la mayor potencia que pueden ofrecer las CPU's multi-core. Suricata está desarrollado para que su gestión e implementación se realicen de forma sencilla. También cuenta con aceleración por GPU (con ciertas limitaciones) y permite una migración fácil a Snort si se desea, ya que emplea las mismas reglas y formatos de salida.

  Aunque no es un producto tan extendido como Snort, cada vez va ganando más adeptos. La aceleración por hardware o la mejora de rendimiento son algunos de los puntos fuertes de esta aplicación [103].

- **OSSEC**

  OSSEC es un IDS que también opera a su vez como un gestor de incidencias de seguridad. Es un software altamente adaptable gracias a sus diversas opciones de configuración, posibilidad de habilitación de alertas personalizadas y escritura de reglas de acciones en respuesta a las alertas de seguridad detectadas.

  El proyecto se liberó en 2008 y poco después fue adquirido por la compañía Third Brigade, que se comprometió a seguir manteniendo el código de OSSEC abierto para que toda la comunidad pudiese contribuir a su mejora. Actualmente cuenta con una comunidad de desarrolladores y colaboradores bastante sólida.

  Volviendo a poner el foco en las características del IDS, éste consta de una aplicación principal (que soporta múltiples sistemas operativos), una interfaz web, la cual proporciona una interfaz gráfica para el usuario y, por último, un agente especial para Windows, facilitado para cuando se instala OSSEC en entornos con este SO. OSSEC permite que la captura del tráfico de red se realice a través de estos agentes, pero también permite prescindir de ellos, posibilitando la monitorización de la red a través de firewalls o routers. Otra funcionalidad destacable es que realiza la comprobación de integridad de los ficheros del sistema, protegiéndolos así de modificaciones no autorizadas. Relativo a las notificaciones que proporciona el IDS, éstas se pueden personalizar, y pueden ser integradas con SMTP u otros para ser enviadas por correo electrónico a ciertos dispositivos [105].

II. Dispositivo sonda

El dispositivo sonda o sniffer se encarga en primer lugar de capturar y preprocesar el tráfico de red en bruto; posteriormente, el IDS (que suele encontrarse instalado en el mismo dispositivo sonda) compara dicho tráfico de red con las reglas de seguridad predefinidas, emitiendo alertas que informan al responsable de seguridad de eventos sospechosos detectados. El hardware elegido para hacer las veces de dispositivo sonda puede ser diverso, desde una Raspberry Pi si se quiere buscar una solución económica y que permita la distribución de muchas sondas por la red, hasta equipos mucho más potentes, que realizan las acciones del dispositivo sonda y las del servidor final [103].

El número de dispositivos y su localización en la red dan lugar a diversas configuraciones, cada una de ellas con sus ventajas y desventajas. Cada una de las ubicaciones presentadas a continuación no es excluyente, y la configuración final dependerá de los requerimientos del cliente. La Figura 52 muestra un pequeño escenario que ejemplifica la red de una empresa, y que servirá para especificar a continuación las distintas ubicaciones que puede tener un dispositivo sonda en la red [102].

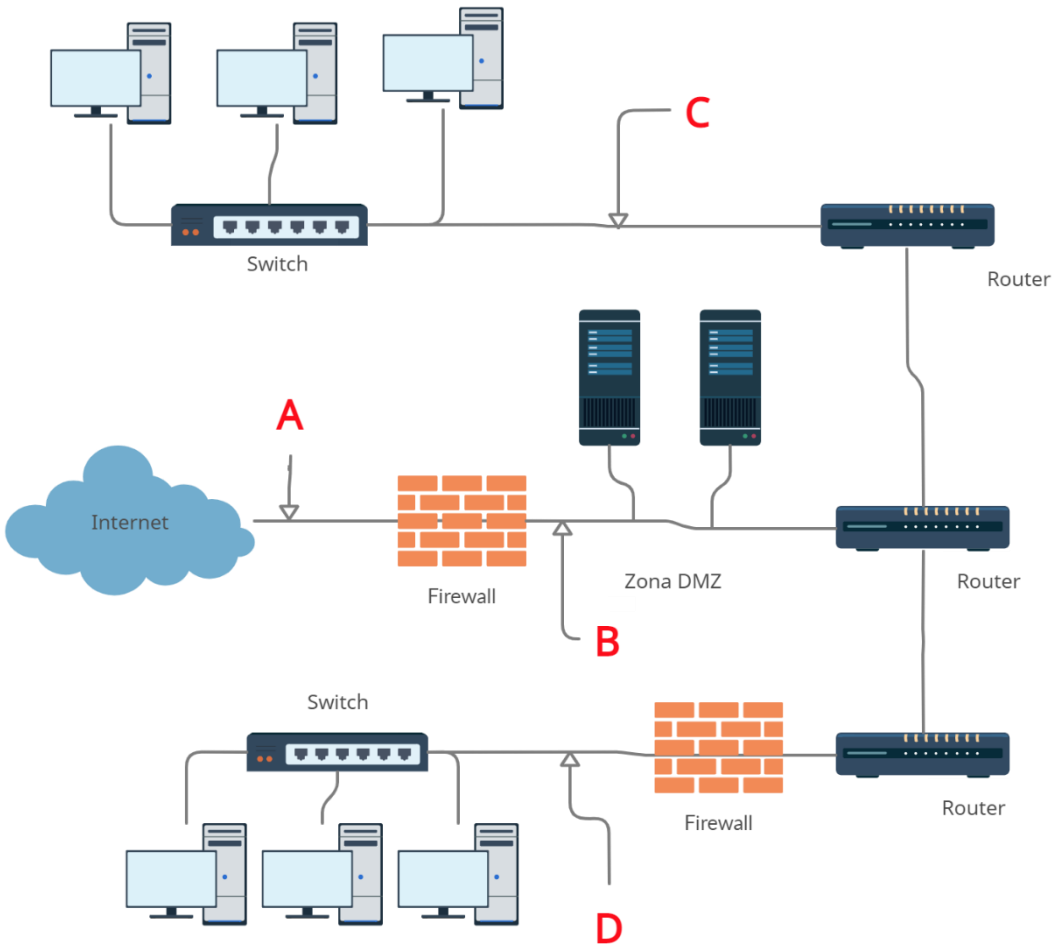


Figura 68: Ejemplo de diagrama de red de una empresa pequeña.

- **Dispositivo sonda delante del cortafuegos externo (A)**
  Colocar el dispositivo en esta ubicación tiene como ventajas la monitorización del número y tipología de ataques que recibe la empresa, así como la detección de ataques cuyo objetivo principal es el firewall externo.

  En contraposición, no puede detectar ataques que utilicen algún tipo de encriptado para ocultar la información. Asimismo, si está mal configurado puede saturarse, ya que en este punto el flujo de datos suele ser bastante abundante. Por último, si el IDS es detectado por algún atacante puede convertirse en un blanco fácil, dejando así desprotegido al sistema [102].

- **Dispositivo sonda detrás del cortafuegos externo (B)**
  En este caso, el dispositivo sonda se encuentra implementado en la zona desmilitarizada o zona DMZ[24]. En este punto de la red suelen encontrarse los servicios principales del sistema (servidores HTTP, FTP, DNS…). Comúnmente, está protegida por firewalls y otros elementos. Este posicionamiento permite detectar ataques que consiguen atravesar el firewall externo, así como aquellos contra los servidores que ofrecen los servicios públicos situados en la zona DMZ. Si no es capaz de detectar algunos ataques, sí que puede identificar el rastro que pueden dejar, como intentos de conexiones salientes. Además, la detección de estos ataques en esta zona permite mejorar la configuración del cortafuegos externo, para evitar que estos ataques vuelvan a penetrar en el sistema.

  Por el contrario, esta ubicación no permite identificar ataques que utilicen métodos de encriptación. En este tramo del sistema también existe un gran flujo de datos, por lo que el IDS debe estar correctamente configurado para que no descarte el tráfico relevante. En último lugar, es necesario tomar medidas adicionales para asegurar el IDS, a pesar de que se encuentre detrás del cortafuegos [102].

- **Dispositivo sonda monitorizando redes principales (C)**
  Esta localización monitoriza las redes del sistema con mayor actividad. Al estar en una posición donde existe un gran flujo de información, también hay mayor probabilidad de detectar ataques. Además, se pueden detectar ataques internos, realizados por el propio personal de la empresa. Por el contrario, esta posición hace que el IDS sea muy vulnerable a ataques internos, ni tampoco se pueden evitar problemas relacionados asociados a los conmutadores. Como en las anteriores posiciones, también tiene problemas con ataques encriptados [102].

- **Dispositivo sonda monitorizando subredes críticas (D)**
  En ocasiones, los servidores y recursos más importantes de una empresa son situados en una subred privada, separada de la red principal mediante dispositivos como firewalls. Suele ser recomendable situar IDS en estas subredes para proteger estas subredes pertinentemente. Sin embargo, al estar colocados en esta posición, no están estratégicamente bien posicionados para detectar ataques de origen interno.

III. Recolección de alertas

Este software se encarga de centralizar los logs generados por el IDS y enviarlos a la base de datos que se encarga de almacenarlos. Las opciones más extendidas son [103]:

- **Rsyslog**
  Rsyslog es una utilidad software de código abierto para enviar logs en una red con sistemas informáticos UNIX o similares. Implementa el protocolo syslog básico y lo extiende con capacidades de filtrado, opciones de configuración flexibles y añade nuevas funcionalidades como el uso de TCP para el transporte de las alertas. Por el contrario, las opciones de envío no admiten muchas modificaciones.

[24] *Para más información sobre la zona DMZ, ir al Capítulo 1.*

- **Logstash**
  Logstash puede unificar bajo un solo formato datos recibidos de fuentes diferentes, y normalizar los datos en destinos de su elección. Cualquier evento recibido puede ser modificado con una gran variedad de plugins y filtros, para enriquecer el evento con toda la información que resulte de importancia. Puede ser ejecutada en cualquier SO, debido a que está basada en jRuby (requiere de una Java Virtual Machine para funcionar). Otra gran ventaja es que este software está diseñado por los creadores de la base de datos ElasticSearch, por lo que los problemas de compatibilidad que pudiese haber quedan automáticamente descartados si se usa este recolector de alertas junto con ElasticSearch.

- **Beats**
  Beats es una recopilación de agentes de reenvío de código abierto, desarrollados también por los creadores de ElasticSearch. Los distintos agentes envían los logs, métricas y demás a ElasticSearch. Algunos de estos agentes son Packetbeat (analizador de paquetes de red que envía información sobre las transacciones intercambiadas entre sus servidores de aplicaciones), Filebeat (envía archivos de registro) o Metricbeat (recopila métricas de los SO y servicios ejecutados en los servidores).

IV. Base de datos

Esta base de datos será la que almacene todas las alertas que le llegan desde el recolector y en última instancia interactuará con la interfaz gráfica para mostrar los eventos al responsable de seguridad. Algunas de las bases de datos usadas para soluciones IDS son:

- **ElasticSearch**
  Esta base de datos es la solución más utilizada de código abierto. Está basado en Lucene, una API pensada para recuperación de información. Gracias a esto, ElasticSearch ofrece capacidades de búsqueda de texto, autocompletado, etc. Entre sus características más destacadas se cuenta la indexación de grandes cantidades de datos, para poder ser consultados posteriormente. Además, es muy escalable: el software se organiza en nodos, los cuales son alojados dentro de un cluster. Se pueden añadir nuevos nodos al cluster, por ejemplo, por si hay alguno que esté fallando poder reorganizar la información y hacer que los datos siempre estén disponibles [106].

- **MySQL, PostgreSQL**
  Estas bases de datos relacionales son recomendadas si se usa Snort como sistema de detección de intrusos. Ambas bases de datos son de código abierto; sin embargo, MySQL está más orientado a bases de datos pequeñas y medianas. Si se necesita un uso sencillo, esta sería la mejor opción. Para bases de datos mucho más grandes y con consultas frecuentes y más largas es mucho mejor elección usar PostgreSQL [107].

V. Interfaz gráfica

Por último, la interfaz gráfica resulta fundamental para facilitar la interpretación de los eventos de seguridad detectados. A mayor claridad y posibilidad de filtrar los datos según distintos parámetros, más eficiente será la labor de investigación realizada por el empleado de seguridad. Algunas de las interfaces gráficas de visualización de eventos son [103]:

- **Graylog**

  Este software permite la visualización de registros almacenados en una base de datos ElasticSearch, además de la realización de consultas avanzadas sobre dichos eventos, crear tablones de visualización con los resultados de estas consultas e incluso generar alarmas. La parte cliente de Graylog se maneja a través de una interfaz web y dispone de acceso con usuario y contraseña para gestionar los accesos a la aplicación. La parte servidor solo funcionará sobre plataformas Linux.

  Se necesita además de una base de datos adicional MongoDB o MariaDB para guardar todos los datos relativos a la configuración de la aplicación, usuarios que pueden acceder al sistema, etc.

- **Kibana**

  Kibana se posiciona como una alternativa a Graylog para bases de datos ElasticSearch. El enfoque de esta herramienta está centrado en el Big Data, por lo que la visualización de los eventos es muy variada: mapas de calor, localizaciones de los eventos, etc. El acceso al cliente se realiza vía web, y la parte servidor no tiene restricciones de sistema operativo a diferencia de Graylog, ya que se monta sobre una Java Virtual Machine. Además, la integración con todo el ecosistema de ElasticSearch está garantizada, al estar recomendada y creada por los fabricantes de esta base de datos.

- **Splunk**

  La única alternativa que se propone en esta relación de herramientas que es de pago es el software Splunk. Esta herramienta potentísima permite recopilar, almacenar y correlacionar datos en tiempo real. Estos datos pueden provenir de diversas ubicaciones, como una red, un sistema concreto, etc. Splunk recolecta los logs recogidos por sus agentes, y los guarda en el servidor principal. Una vez están en este punto, dichos eventos pueden ser tratados para detectar ataques y aplicar las correcciones necesarias para evitar que un ataque sea efectivo o minimizar daños.

  Esta herramienta consta de una interfaz web muy sencilla de manejar, que ofrece la posibilidad de revisar los logs almacenados, crear paneles con los resultados, crear reportes o generar alertas. Además, resulta ser muy flexible y escalable.

  Además del módulo principal Splunk Enterprise, se pueden añadir otros módulos de pago que incrementan las funcionalidades del software, proporcionando herramientas definidas para la búsqueda de comportamientos maliciosos [108].

6.2 Implementación del sistema completo

Llegados a este punto, se han comentado de forma somera las consideraciones a tener en cuenta para el desarrollo de un software IDS y se han expuesto todos los elementos que integran un sistema de detección de intrusos. Una vez se ha desarrollado todo el proceso previo de planificación, preparación y formación especializada del personal de los administradores del sistema (que puede que no dispongan de todo el conocimiento necesario para explotar el IDS al máximo) llega el momento de implementar la solución final en el sistema.

Generalmente, la solución instalada en la red suele ser un compendio de sistemas de detección de intrusos basados en red (NIDS) y sistemas de detección de intrusos basados en máquina (HIDS). Si se realiza la implementación de forma gradual, los propios administradores irán adquiriendo más experiencia en el sistema con cada integración de un nuevo IDS en la red. Si se adopta esta opción, primero se comienza con la instalación de los NIDS, más fáciles de gestionar, finalizando con la instalación de los HIDS en los equipos críticos. Adicionalmente, la implementación de un IDS debe completarse con el uso periódico de analizadores de vulnerabilidades sobre los IDS y otros elementos de seguridad, así se ayuda a mantener la confianza sobre estos dispositivos [103].

Una tarea recomendable para reforzar el éxito de un sistema de estas características es el desarrollo de unas políticas de seguridad acordes con las necesidades de la organización. Cuando se crea una política de seguridad, es necesario que incluya por qué ésta es importante para la empresa, y debe estar concebida para conducir a la ejecución de ciertos procedimientos una vez se detectan señales de que se ha producido una intrusión. Para desarrollar una política de seguridad de forma correcta, se recomienda cubrir las siguientes etapas [109]:

- La introducción y propósito sientan las bases del resto del documento, tratando de forma global el contenido de éste, y explicando por qué es necesaria esta tecnología y esta política en la organización.
- El alcance es muy importante, puesto que define a quién afecta la política de seguridad: puede tratarse de un departamento en concreto, de toda la empresa, de un reducido grupo de personas, etc.
- La parte de políticas constituye el cuerpo del documento, aclarando sobre cómo actuar al ocurrir determinadas situaciones.
- La aplicación de la ley define las consecuencias a aplicar a las personas mencionadas en el apartado de alcance. Se puede incluir en este punto algún tipo de manual de recursos humanos, o similar.
- Por último, se encuentran las secciones de definiciones y revisiones. La primera recoge terminología que puede resultar de utilidad a la hora de comprender totalmente todos los conceptos que se recogen en el documento, mientras que la segunda refleja y recoge los distintos cambios que va sufriendo la política a lo largo del tiempo [109].

Ya definida la política de seguridad a seguir por la empresa, se comienza a implementar el sistema de detección de intrusos en la red. Se instala y configura el sistema operativo tanto del dispositivo sonda como del servidor final, configurando también las interfaces de red correctamente. Posteriormente, se instala y configura el IDS en el dispositivo sonda: hay que especificar la interfaz o interfaces de red que se desean monitorizar, añadir (si se desean) nuevas condiciones para detectar tráfico malicioso, etc. Por último, se instala y configura la base de datos, el recolector de alertas y la interfaz gráfica en el servidor.

Como paso final de todo el proceso, para comprobar que toda la infraestructura se ha desplegado correctamente y que el funcionamiento es el deseado, se realizan pruebas controladas lanzando ataques, tanto conocidos por el IDS como enmascarados. Por citar un caso que podría darse, para la comprobación de las pruebas con ataques estándar éstos podrían lanzarse desde una máquina externa a

la organización durante varios días. Si los ataques han sido recogidos en la base de datos del servidor final, es que la configuración del sistema funciona según lo esperado. Por otro lado, para enmascarar los ataques y realizar pruebas con esta configuración se puede usar un software denominado Fragroute, que modifica los paquetes TCP/IP enviados por la red de diversas maneras: los puede fragmentar o desordenar, entre otras acciones [110].

Particularmente, si se quisiese implementar un sistema de detección de intrusos en una red privada como la red inalámbrica de una vivienda, probablemente la mejor opción sea utilizar un sistema de detección de intrusos basado en máquina. Cualquier software IDS de los presentados anteriormente puede implementarse de esta forma, siempre que se pueda instalar el agente del IDS en la propia máquina. Tener un IDS basado en máquina permite superar la dificultad del tráfico cifrado, ya que los paquetes se desencriptan al llegar al destino y el IDS ve el tráfico descifrado; no obstante, consume recursos de la propia máquina que monitoriza [102].

Esta sería en líneas generales, la hoja de ruta a seguir si se quisiese implementar realmente un sistema de detección de intrusos en una red empresarial. Aun sin entrar en detalles minuciosos sobre el tema, se ha ofrecido una idea global sobre cómo materializar un sistema de este tipo desde el punto de vista de un cliente que debe cubrir sus necesidades de seguridad con un IDS.

Como cierre de este Trabajo Fin de Máster, en el próximo capítulo se presentan las líneas de desarrollo que pueden tomar los IDS próximamente, así como comentarios de posibles ampliaciones y mejoras que pueden derivarse del trabajo desarrollado en el presente documento. Además, a la vista de las conclusiones obtenidas de forma individual para los algoritmos de aprendizaje máquina y aprendizaje profundo, se aporta una recapitulación global de todas ellas.

7 CONCLUSIONES Y LÍNEAS DE FUTURO

Como epílogo de este trabajo, es de obligado cumplimiento hacer referencia a los caminos que tomará la investigación relacionada con la IA y los IDS. Como se ha podido constatar a lo largo de los capítulos, el desarrollo de los algoritmos de machine learning y deep learning para sistemas de detección de intrusos es un tema de largo recorrido. Las investigaciones sobre todo lo relacionado con la inteligencia artificial proliferan en la actualidad, y de las mejoras conseguidas en ellas se benefician los sistemas de detección de intrusos. Se comenzará el capítulo enunciando un resumen de conclusiones globales y comentarios acerca de los resultados obtenidos en el trabajo o de distintos aspectos que resultan destacables, aportando distintas opciones de mejora para una posible continuación del trabajo aquí comenzado. Por último, se expondrán los planes de futuro que se avistan para los algoritmos empleados en sistemas de detección de intrusos.

7.1 Conclusiones globales y líneas futuras relativas a la optimización del trabajo

En vista de los resultados obtenidos a lo largo del desarrollo de este documento, existen sobre todo dos conclusiones principales sobre las que poner el foco: la problemática ocasionada por la UNSW-NB15 y la dificultad de clasificar del CVAE, siendo superado por otros algoritmos de *machine learning*. De estas dos claves surgen posibles vías a explorar para intentar subsanar los problemas detectados

- **Dificultad de la UNSW-NB15:** Una de las conclusiones más evidentes del desarrollo del trabajo es la dificultad que entraña la base de datos elegida sobre todo en lo que a clasificación multiclase se refiere. En primer lugar, que el conjunto de datos esté bastante imbalanceado puede ser una causa de los problemas sufridos no solo por el CVAE, si no también en el resto de modelos, al realizar clasificación multiclase. Todos los modelos desarrollados sufrían en la clasificación de las clases minoritarias (por ejemplo, *shellcode* ó DoS), por lo que resultaría interesante aplicar al conjunto de datos un algoritmo de generación de muestras sintéticas como SMOTE, para igualar en muestras todas las clases y ver si los modelos mejoran de esta forma su rendimiento global. Incluso el mismo autocodificador variacional podría ser utilizado para la generación de estas muestras sintéticas. Además, se ha constatado de forma empírica que, a mayor número de muestras de una clase, mejor resultado arroja el modelo a la hora de clasificar. Explorar y preprocesar la base de datos con mayor detalle (selección de características, eliminación de caracteres extraños e instancias redundantes, etc.) probablemente tendría un impacto positivo en el desempeño en tareas clasificatorias de los modelos.

- **Selección de características en la base de datos:** Teniendo en cuenta que uno de los mayores problemas de los algoritmos de deep learning reside en su gran consumo computacional y temporal, la selección de características en la base de datos es una vía a explorar para recortar el tiempo y complejidad computacional del modelo, teniendo éste que procesar un menor número de características (procesando solamente las realmente importantes y decisivas en la clasificación).

En el presente trabajo se intentó emplear WEKA para aplicar selección de características a la UNSW-NB15 sin éxito debido a la complejidad computacional que tenía el algoritmo, lo que se traducía en una cantidad de tiempo considerable de ejecución; además, se realizó solamente para uno de los casos del algoritmo *random forest* y para un conjunto pequeño de instancias. Sería interesante aplicar distintos métodos de selección de características a la base de datos y ver cuál de ellos arroja mejor resultado, así como introducir esta base de datos con selección de características aplicada en el autocodificador variacional condicional.

- **Preferencia de otros algoritmos como clasificadores en lugar del CVAE:** La comparativa de resultados entre los tres tipos de algoritmos distintos demuestra que el autocodificador variacional no obtiene buenos resultados a la hora de trabajar como clasificador, y resulta preferible confiar en otros modelos, como por ejemplo el *random forest*. Este último es el algoritmo que mejor ha sido capaz de clasificar la base de datos UNSW-NB15 de los tres.

- **Optimización del CVAE para IDS:** Todavía queda trabajo por delante en la investigación de algoritmos de DL para sistemas de detección de intrusos, concretamente en lo respectivo a los autocodificadores variacionales. Los estudios que utilizan este modelo están muy enfocados en el procesamiento de imágenes, pero para tareas clasificatorias de ataques son pocos y muy recientes. En combinación con lo mencionado en los anteriores apartados, se podrían cambiar diversos aspectos del modelo (epochs, estructura de capas, parámetros del optimizador…) viendo si afectan positivamente a los resultados o no.

- **Combinación de algoritmos:** Otra opción es la de usar el autocodificador variacional en conjunto con otros modelos para mejorar los resultados de clasificación. De hecho, recientes estudios plantean la posibilidad de elaborar algoritmos híbridos, concatenando distintos algoritmos. Por ejemplo, en 2019 Yanquing Yang y Kangfeng Zheng, entre otros, propusieron un modelo que lleva a cabo precisamente esta idea. En este artículo, se utiliza un CVAE para aprender la distribución del conjunto de datos y generar nuevas muestras que permiten el balanceo de la base de datos, mejorando así el nivel de detección de las clases minoritarias; posteriormente, una red neuronal profunda es la encargada de extraer las características de alto nivel del dataset, ajustar automáticamente los pesos de la red y, en última instancia, efectuar la clasificación de las muestras en las distintas clases [111]. Hay que tener en cuenta que una de las aplicaciones principales del autocodificador variacional es la de generación de muestras sintéticas, por lo que usarlo para este tipo de tareas y utilizar otro algoritmo que se encargue de la clasificación puede mejorar el rendimiento global del modelo al completo.

7.2 Líneas futuras relativas al desarrollo de los algoritmos para sistemas de detección de intrusos

En lo referente a las vías de investigación próximas en el campo de los algoritmos para IDS, algunas de ellas reflejan problemas que se han vislumbrado en el desarrollo del presente trabajo y que se han propuesto en el apartado anterior. Entre las ideas de futuro propuestas para la evolución de los IDS, también se cuentan el desarrollo de las bases de datos utilizadas para entrenar los IDS, así como algunas ideas más concretas para tecnologías particulares como IoT. Se destacan las siguientes tendencias de futuro y desafíos que deberá afrontar la investigación en este ámbito [112]:

- **Asegurar la eficiencia de un sistema de detección de intrusiones completo:** Recientes estudios han puesto de manifiesto la debilidad de los IDS contra ataques "día-cero", devolviendo un gran número de falsas alarmas. Para contrarrestar o minimizar las falsas alarmas devueltas por los sistemas de detección de intrusos en estos casos, resulta esencial contar con un conjunto de datos actualizado y equilibrado para mejorar el rendimiento del sistema. En la actualidad todavía no se ha definido un marco eficiente y completo de los IDS para proporcionar completa seguridad ante cualquier tipo de ataque. Se torna necesaria la definición de un marco que incluya un mecanismo para actualizar las definiciones de ataques del IDS de forma periódica y entrenar éste de forma rápida, mejorando sustancialmente el modelo y reduciéndose así las falsas alarmas.

- **Algoritmos DL para IDS:** El uso de algoritmos de aprendizaje profundo ha ganado cada vez más popularidad a causa de la capacidad de aprendizaje profundo de las características del conjunto de datos, lo que se traduce en unos resultados muy buenos a la hora de clasificar el tráfico entre normal y anómalo. No obstante (y como ya se ha venido destacando anteriormente), estos algoritmos son bastante complejos y necesitan de altas capacidades computacionales, de almacenamiento y temporales. Las GPU de alto rendimiento dan solución a estos problemas, pero suelen ser muy caras. Por tanto, hay que alcanzar un equilibrio entre rendimiento y coste. Una solución plausible para intentar reducir el coste económico podría pasar por ofrecer servicios de GPU en la nube; otra alternativa es intentar reducir la complejidad de los algoritmos de aprendizaje profundo realizando selección de características de forma eficiente e inteligente, para así usar menos recursos computacionales y conseguir prácticamente el mismo rendimiento que trabajando con el conjunto completo de características. Asimismo, también se puede desarrollar la idea de usar deep learning para selección de características y manipulación del conjunto de datos y machine learning para la clasificación, reduciéndose así también la complejidad del modelo propuesto. En definitiva, el deep learning todavía tiene un enorme potencial por explotar a la hora de ser aplicado en IDS.

- **Aplicación de IDS en sistemas ciberfísicos:** Actualmente existe un interés importante en los sistemas SCADA debido a la incipiente automatización de procesos industriales, así como en las redes de comunicaciones para vehículos aéreos no tripulados (UAV). En el caso de las primeras, su complejidad va en aumento a medida que se incorporan a ellas las últimas tecnologías de comunicaciones existentes. Esto ofrece una oportunidad a los ciberatacantes de vulnerar la integridad de estos sistemas. Por esta razón, los IDS toman un papel vital en este tipo de sistemas, para evitar y detectar a tiempo ataques que podrían ocasionar problemas en el sistema que controla el SCADA. En contraposición, los IDS dedicados a sistemas de este tipo no están aún muy estudiados, por lo que se requiere continuar la investigación en este ámbito para poder diseñar sistemas de detección de intrusos eficientes en estos sistemas.
  Del mismo modo, los UAV se utilizan en aplicaciones como vigilancia de infraestructuras críticas o del tráfico, entre otros. Debido a que la comunicación se realiza a través del canal inalámbrico, esta red está accesible para que los ciberatacantes intenten lanzar ataques. Entonces se requiere un IDS eficiente e inteligente que sea capaz de detectar a los intrusos que se adentran en las redes habilitadas por los UAV. De la misma forma que ocurría con los sistemas SCADA, esta idea aún está lejos de ser aplicable de inmediato, y precisa de más investigación.

- **Falta de una base de datos actualizada:** Las bases de datos existentes no cuentan con los ataques más recientes en su conjunto de datos, por lo que la gran mayoría de algoritmos fallan sobre todo a la hora de detectar ataques "zero-day", como ya se expuso en líneas anteriores. Para lograr un

modelo IDS eficiente y con un rendimiento óptimo hace falta probarlo y verificarlo usando un conjunto de datos que disponga de todo tipo de ataques, incluidos los más recientes. Así, el modelo puede aprender más patrones y proporcionará protección contra las intrusiones más nuevas. Sin embargo, la construcción sistemática de un conjunto de datos con muestras suficientes y actualizadas de los ataques no es una tarea sencilla y requiere de muchos recursos. Es por eso que este tema se convierte en uno de los desafíos de investigación más importantes para los IDS.

- **Baja precisión debido a las bases de datos imbalanceadas:** Este ha sido uno de los problemas sufridos con el conjunto de datos UNSW-NB15. A causa del desequilibrio existente entre las distintas clases de los conjuntos de datos, hay ataques que arrojan una menor precisión de detección que la precisión de detección general del modelo (concretamente, aquellas clases con menor número de instancias). Para solucionarlo existen dos alternativas: bien conseguir un conjunto de datos actualizados y equilibrados, o generar de forma sintética muestras parecidas a las que conforman las clases minoritarias para equilibrar el número de instancias de todas las clases. Ya existen algoritmos creados específicamente para estos menesteres como el SMOTE o el ADASYN, pero el margen de mejora en este problema sigue siendo significativo.

- **Rendimiento pobre de los algoritmos en entorno real:** La mayoría de los estudios desarrollados se testan y verifican en un laboratorio, usando las bases de datos relativas a sistemas de detección de intrusos públicas. Entonces, a pesar de que dichas pruebas cosechen resultados positivos, no está claro si el comportamiento será el mismo en el mundo real. La metodología de trabajo propuesta debería contemplar las pruebas del algoritmo en un entorno real después de las pruebas controladas en un laboratorio, para verificar así su comportamiento en las redes actuales.

- **IDS adaptado a IoT:** El IoT cada vez prolifera más en nuestras infraestructuras y sociedad, por lo cual es un paso lógico que los IDS también proporcionen seguridad a estas redes y sus sensores asociados. Estos sensores recogen una enorme cantidad de datos críticos que se centralizan en un servidor, y que después viajan a través de la red. El sistema de detección de intrusos puede desplegarse en los puntos donde el tráfico entra desde internet a la red IoT, o bien de forma distribuida en los nodos sensores. En el primer caso, el IDS necesita ser eficiente hasta en los ataques más recientes, mientras que en el segundo caso se necesitan algoritmos de IDS ligeros, puesto que los nodos sensores suelen ser dispositivos con recursos limitados: relativamente poca capacidad computacional y almacenamiento. Por tanto, hay que conseguir un equilibrio en términos de potencia de cálculo, tiempo de entrenamiento y rendimiento del algoritmo.

8 BIBLIOGRAFÍA

[1] M. Valle, "Por qué la ciberseguridad es más necesaria que nunca.," *Bit Life Media*, 2020. https://bitlifemedia.com/2020/03/por-que-la-ciberseguridad-es-mas-necesaria-que-nunca/.

[2] M. Inc., "2019 Global Risk Perception Survey," *Microsoft*, 2019. https://www.microsoft.com/security/blog/wp-content/uploads/2019/09/Marsh-Microsoft-2019-Global-Cyber-Risk-Perception-Survey.pdf.

[3] EY, "Cybersecurity and the Internet of Things.," *EY*, 2015. https://www.ey.com/Publication/vwLUAssets/EY-cybersecurity-and-the-internet-of-things/%24FILE/EY-cybersecurity-and-the-internet-of-things.pdf.

[4] M. Bayern, "Kaspersky honeypots find 105 million attacks on IoT devices in first half of 2019.," *TechRepublic*, 2019, [Online]. Available: https://www.techrepublic.com/article/kaspersky-honeypots-find-105-million-attacks-on-iot-devices-in-first-half-of-2019/.

[5] J. De la Hoz, Emiro;De la Hoz, Eduardo; Ortiz, Andrés; Ortega, "Modelo de detección de intrusiones en sistemas de red, realizando selección de características con FDR y entrenamiento y clasificación con SOM," *Inge-Cuc*, vol. 8, pp. 85–116, 2012.

[6] D. Daniels, "14 Network Security Tools and Techniques to Know.," *Gigamon*, 2019. https://blog.gigamon.com/2019/06/13/what-is-network-security-14-tools-and-techniques-to-know/.

[7] E. de .J., "Deep Machine Learning," *JTSEC*, 2017. .

[8] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, 2019, doi: 10.1186/s42400-019-0038-7.

[9] A. Drewek-Ossowicka, M. Pietrołaj, and J. Rumiński, "A survey of neural networks usage for intrusion detection systems," *J. Ambient Intell. Humaniz. Comput.*, 2020, doi: 10.1007/s12652-020-02014-x.

[10] C. C. Urcuqui L., M. Garcia P., and J. L. Osorio Q., *Ciberseguridad: un enfoque desde la ciencia de datos*. Editorial Universidad Icesi, 2018.

[11] "What is Cyber Security?," *Kaspersky*, 2020. https://www.kaspersky.com/resource-center/definitions/what-is-cyber-security.

[12] J. L. Harrington, "Chapter 1 - In the Beginning …," in *The Morgan Kaufmann Series in Networking*, J. L. B. T.-N. S. Harrington, Ed. San Francisco: Morgan Kaufmann, 2005, pp. 1–34.

[13] D. Sutherland, "Wired vs Wireless Networking.," *Aberdeen Cyber Security*, 2019. https://aberdeencybersecurity.co.uk/wired-vs-wireless-networking/.

[14] J. L. Harrington, "Chapter 2 - Basic Security Architecture," in *The Morgan Kaufmann Series in Networking*, J. L. B. T.-N. S. Harrington, Ed. San Francisco: Morgan Kaufmann, 2005, pp. 35–53.

[15] INCIBE, "Qué es una DMZ y cómo te puede ayudar a proteger tu empresa.," *INCIBE*, 2019. https://www.incibe.es/protege-tu-empresa/blog/dmz-y-te-puede-ayudar-proteger-tu-empresa.

[16] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, "Network attacks: Taxonomy, tools and systems," *J. Netw. Comput. Appl.*, vol. 40, pp. 307–324, 2014, doi: https://doi.org/10.1016/j.jnca.2013.08.001.

[17] "Types of Cyber Attacks," *Javatpoint*, 2018. https://www.javatpoint.com/types-of-cyber-attacks.

[18] "Web Browser-Based Attacks," *Morphisec*, 2019. https://www.morphisec.com/hubfs/1111/briefs/BrowserAttacksBrief-190327.pdf.

[19] Avast, "Inyección SQL," *Avast*, 2020. https://www.avast.com/es-es/c-sql-injection.

[20] J. Jiménez, "Man in The Middle: qué son estos ataques y cómo prevenirlos.," *Redeszone*, 2020. https://www.redeszone.net/tutoriales/seguridad/ataques-man-in-the-middle-evitar/.

[21] "¿Qué es un ataque de fuerza bruta?," *Kaspersky*, 2018. https://www.kaspersky.es/resource-center/definitions/brute-force-attack.

[22] I. Pérez, "Comprendiendo la vulnerabilidad XSS (Cross-site Scripting) en sitios web.," *We Live Security*, 2015. https://www.welivesecurity.com/la-es/2015/04/29/vulnerabilidad-xss-cross-site-scripting-sitios-web/.

[23] "Understanding Denial-of-Service Attacks," *Cybersecurity and Infrastructure Security Agency*, 2019. https://us-cert.cisa.gov/ncas/tips/ST04-015.

[24] "¿Qué es el phishing? Tipos de phishing y ejemplos," *Malwarebytes*, 2019. https://es.malwarebytes.com/phishing/.

[25] M. Rivero, "¿Qué son los virus informáticos?," *Infospyware*, 2011. https://www.infospyware.com/articulos/¿que-son-los-virus-informaticos/#:~:text=Los virus informáticos tienen%2C básicamente,desde una simple broma hasta.

[26] "Gusanos Informáticos," *Panda Security*, 2020. https://www.pandasecurity.com/es/security-info/worm/.

[27] "What Is the Difference: Viruses, Worms, Trojans, and Bots?," *Cisco*, 2014. https://tools.cisco.com/security/center/resources/virus_differences#:~:text=Computer worms are similar to,or human help to propagate.

[28] J. Albors, "¿Sabes qué es un exploit y cómo funciona?," *We Live Security*, 2015. https://www.welivesecurity.com/la-es/2014/10/09/exploits-que-son-como-funcionan/.

[29] N. Moustafa and J. Slay, "The evaluation of Network Anomaly Detection Systems: Statistical analysis of the UNSW-NB15 data set and the comparison with the KDD99 data set," *Inf. Secur. J.*, 2016, doi: 10.1080/19393555.2015.1125974.

[30] "Sistemas de detección y prevención IDS e IPS: ¿Para qué sirven?," *Punt Informàtic*, 2017. https://puntinformatic.com/sistemas-de-deteccion-y-prevencion-ids-e-ips/.

[31] H. Alaidaros, M. Mahmuddin, and a L. I. a L. Mazari, "An overview of flow-based and packet-based intrusion detection performance in high speed networks," *Proc. Int. Arab Conf. Inf. Technol.*, 2011.

[32] "Sistema de prevención de intrusos," *Wikipedia*, 2020. https://es.wikipedia.org/wiki/Sistema_de_prevención_de_intrusos.

[33] IEEE Computer Society, "The Impact of AI on Cybersecurity," *IEEE Computer Society*, 2020. https://www.computer.org/publications/tech-news/trends/the-impact-of-ai-on-cybersecurity/.

[34] P. Ongsulee, "Artificial intelligence, machine learning and deep learning," in *International Conference on ICT and Knowledge Engineering*, 2018, doi: 10.1109/ICTKE.2017.8259629.

[35] J. J. Bryson, "La última década y el futuro del impacto de la IA en la sociedad.," *OpenMind*, 2020. https://www.bbvaopenmind.com/articulos/la-ultima-decada-y-el-futuro-del-impacto-de-la-ia-en-la-sociedad/.

[36] F. Chollet, *Deep Learning with Python*. 2017.

[37] E. Alpaydin, "Introduction to Machine Learning," *Introd. to Mach. Learn. Third Ed.*, 2014.

[38] Mathworks, "Deep Learning: Tres cosas que es necesario saber," *Mathworks*, 2016. https://es.mathworks.com/discovery/deep-learning.html.

[39] M. Chatterjee, "Top 20 Applications of Deep Learning in 2020 Across Industries," *www.greatlearning.com*, 2020. https://www.mygreatlearning.com/blog/deep-learning-applications/#healthcare.

[40] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. 2017.

[41] M. Merino, "Conceptos de inteligencia artificial: qué es el aprendizaje por refuerzo," *Xataka*, 2019. https://www.xataka.com/inteligencia-artificial/conceptos-inteligencia-artificial-que-aprendizaje-refuerzo.

[42] H. Liu and B. Lang, "Machine learning and deep learning methods for intrusion detection systems: A survey," *Applied Sciences (Switzerland)*. 2019, doi: 10.3390/app9204396.

[43] "Naive Bayes," *TechShayari*, 2019. https://techshayari.wordpress.com/2019/02/12/naive-bayes/.

[44] O. Harrison, "Machine Learning Basics with the K-Nearest Neighbors Algorithm.," *Medium*, 2019. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[45] S. Patel, "Chapter 2 : SVM (Support Vector Machine) — Theory - Machine Learning," *Medium*, 2018. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72.

[46] "Machine Learning y Support Vector Machines: porque el tiempo es dinero.," *Analítica web*, 2016. https://www.analiticaweb.es/machine-learning-y-support-vector-machines-porque-el-tiempo-es-dinero-2/.

[47] R. S. Brid, "Decision Trees — A simple way to visualize a decision," *Medium*, 2018. https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb.

[48] "Decision Tree Full Course | #5. Chi-Square to Select the Best Split Point in Decision Trees.," *Analytics Vidhya*, 2020. https://www.youtube.com/watch?time_continue=283&v=8cwwewynQ6k&feature=emb_logo.

[49] F. J. Payán Somet, "Tema 1: Información y Entropía," in *Apuntes de la asignatura "Tratamiento de la información en comunicaciones digitales,"* 2019.

[50] J. Brownlee, "Information Gain and Mutual Information for Machine Learning," *Machine Learning Mastery*, 2019. https://machinelearningmastery.com/information-gain-and-mutual-information/.

[51] J. Patterson and A. Gibson, *Deep Learning a Practitioner's Approach*. 2019.

[52] F. Arribas Jara, "Aprendizaje no supervisado con modelos generativos profundos," Universidad Autónoma de Madrid, 2018.

[53] "¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador," *Aprende Machine Learning*, 2018. https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[54] "7 pasos del Machine Learning para construir tu máquina.," *Aprende Machine Learning*, 2017. https://www.aprendemachinelearning.com/7-pasos-machine-learning-construir-maquina/.

[55] L. Van Der Maaten, "Accelerating t-SNE using tree-based algorithms," *J. Mach. Learn. Res.*, 2015.

[56] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, 2008.

[57] "Paper Dissected: 'Visualizing Data using t-SNE' Explained," *Machine Learning Explained*, 2018. https://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/.

[58] B. Angelov, "Working with Missing Data in Machine Learning," *Medium*, 2018. https://towardsdatascience.com/working-with-missing-data-in-machine-learning-9c0a430df4ce.

[59] Google Developers, "Imbalanced Data," *Google*, 2020. https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data.

[60] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *J. Big Data*,

2019, doi: 10.1186/s40537-019-0192-5.

[61] "Normalize Data," *Microsoft Docs*, 2019. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data#:~:text=Normalization is a technique often,of values or losing information.

[62] "One Hot Encoding," *Interactive Chaos*, 2019. https://www.interactivechaos.com/manual/tutorial-de-machine-learning/one-hot-encoding.

[63] M. Hauskrecht, R. Pelikan, M. Valko, and J. Lyons-Weiler, "Feature selection and dimensionality reduction in genomics and proteomics," in *Fundamentals of Data Mining in Genomics and Proteomics*, 2007.

[64] Y. Charfaoui, "Hands-on with Feature Selection Techniques: Embedded Methods," *Medium*, 2020. https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-embedded-methods-84747e814dab.

[65] "Sets de Entrenamiento, Test y Validación," *Aprende Machine Learning*, 2020. https://www.aprendemachinelearning.com/sets-de-entrenamiento-test-validacion-cruzada/.

[66] J. Brownlee, "Train-Test Split for Evaluating Machine Learning Algorithms.," *Medium*, 2020. https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/.

[67] M. A. Umar and C. Zhanfang, *Effects of Feature Selection and Normalization on Network Intrusion Detection*. 2020.

[68] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the JAM project," in *Proceedings - DARPA Information Survivability Conference and Exposition, DISCEX 2000*, 2000, doi: 10.1109/DISCEX.2000.821515.

[69] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 2009, doi: 10.1109/CISDA.2009.5356528.

[70] J. Mchugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory," *ACM Trans. Inf. Syst. Secur.*, 2000, doi: 10.1145/382912.382923.

[71] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference, MilCIS 2015 - Proceedings*, 2015, doi: 10.1109/MilCIS.2015.7348942.

[72] C. Kolias, G. Kambourakis, A. Stavrou, and S. Gritzalis, "Intrusion detection in 802.11 networks: Empirical evaluation of threats and a public dataset," *IEEE Commun. Surv. Tutorials*, 2016, doi: 10.1109/COMST.2015.2402161.

[73] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 2018 - Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018, doi: 10.5220/0006639801080116.

[74] A. Gharib, I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "An Evaluation Framework for Intrusion Detection Dataset," in *ICISS 2016 - 2016 International Conference on Information Science and Security*, 2017, doi: 10.1109/ICISSEC.2016.7885840.

[75] R. Panigrahi and S. Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems," *Int. J. Eng. Technol.*, 2018.

[76] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Computers and Security*. 2019, doi: 10.1016/j.cose.2019.06.005.

[77] N. Elmrabit, F. Zhou, F. Li, and H. Zhou, "Evaluation of Machine Learning Algorithms for Anomaly Detection," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 2020, pp. 1–8, doi: 10.1109/CyberSecurity49315.2020.9138871.

[78] N. Moustafa and J. Slay, "The significant features of the UNSW-NB15 and the KDD99 data sets for

Network Intrusion Detection Systems," in *Proceedings - 2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security, BADGERS 2015*, 2017, doi: 10.1109/BADGERS.2015.14.

[79] R. Chapaneri, "Exploratory Analysis of UNSW-NB15 Dataset for Detecting Malicious Network Traffic," *Medium*, 2019. https://medium.com/@radhikachapaneri/exploratory-analysis-of-unsw-nb15-dataset-for-detecting-malicious-network-traffic-6b3e6743e907.

[80] V. Shchutskaya, "Deep Learning: Strengths and Challenges," *InData Labs*, 2019. https://indatalabs.com/blog/deep-learning-strengths-challenges#:~:text=One of deep learning's main,explicitly told to do so.

[81] G. Karatas, O. Demir, and O. K. Sahingoz, "Deep Learning in Intrusion Detection Systems," in *International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism, IBIGDELFT 2018 - Proceedings*, 2019, doi: 10.1109/IBIGDELFT.2018.8625278.

[82] Y. Xin *et al.*, "Machine Learning and Deep Learning Methods for Cybersecurity," *IEEE Access*, 2018, doi: 10.1109/ACCESS.2018.2836950.

[83] R. Sabathe, E. Coutinho, and B. Schuller, "Deep recurrent music writer: Memory-enhanced variational autoencoder-based musical score composition and an objective measure," in *Proceedings of the International Joint Conference on Neural Networks*, 2017, doi: 10.1109/IJCNN.2017.7966292.

[84] E. Karamatli, A. T. Cemgil, and S. Kirbiz, "Audio source separation using variational autoencoders and weak class supervision," *arXiv*. 2018, doi: 10.1109/lsp.2019.2929440.

[85] H. Akrami, A. A. Joshi, J. Li, S. Aydore, and R. M. Leahy, "Brain Lesion Detection Using A Robust Variational Autoencoder and Transfer Learning," in *Proceedings - International Symposium on Biomedical Imaging*, 2020, doi: 10.1109/ISBI45749.2020.9098405.

[86] R. Banerjee and A. Ghose, "A Semi-supervised approach for identifying abnormal heart sounds using variational autoencoder," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2020, doi: 10.1109/ICASSP40776.2020.9054632.

[87] F. S. Caparrini, "Variational AutoEncoder," *Fernando Sancho Caparrini*, 2020. http://www.cs.us.es/~fsancho/?e=232.

[88] J. Jordan, "Variational autoencoders," *Jeremy Jordan*, 2018. https://www.jeremyjordan.me/variational-autoencoders/.

[89] I. Shafkat, "Intuitively Understanding Variational Autoencoders," *Medium*, 2018. https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf.

[90] A. Kristiadi, "Variational Autoencoder: Intuition and Implementation," *Agustinus Kristiadi's blog*, 2016. https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/.

[91] J. Altosaar, "What is a variational autoencoder?," *Jaan Altosar*, 2018. https://jaan.io/what-is-variational-autoencoder-vae-tutorial/.

[92] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

[93] K. Frans, "Variational Autoencoders Explained," *Kevin Frans blog*, 2019. http://kvfrans.com/variational-autoencoders-explained/.

[94] Wikipedia, "Variational Bayesian methods," *Wikipedia*, 2020. https://en.wikipedia.org/wiki/Variational_Bayesian_methods.

[95] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational Inference: A Review for Statisticians," *Journal of the American Statistical Association*. 2017, doi: 10.1080/01621459.2017.1285773.

[96] B. Raj, "Deep Learning," *Carnegie Mellon University*, 2018. http://www.cs.cmu.edu/afs/cs/user/bhiksha/WWW/courses/deeplearning/Fall.2018/www/recitations/rec10.vae.pdf.

[97] "Intuition Behind KL-Divergence Regularization in VAE's," *Deep Learning Course Forums*, 2017.

https://forums.fast.ai/t/intuition-behind-kl-divergence-regularization-in-vaes/1650.

[98] C. Defaux, "Kullback-Leibler Divergence for Machine Learning," *Medium*, 2020. https://medium.com/@cdefaux/kullback-leibler-divergence-for-dummies-c3613bc80ad3.

[99] S. Ruder, "An overview of gradient descent optimization algorithms," *Sebastian Ruder*, 2016. https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants.

[100] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot," *Sensors (Switzerland)*, 2017, doi: 10.3390/s17091967.

[101] S. M. Kasongo and Y. Sun, "Performance Analysis of Intrusion Detection Systems Using a Feature Selection Method on the UNSW-NB15 Dataset," *J. Big Data*, 2020, doi: 10.1186/s40537-020-00379-6.

[102] D. González, "Sistemas de Detección de Intrusiones," *Diego González*, 2003. https://dgonzalez.net/papers/ids/html/.

[103] A. Agra Monte, "Diseño e implementación de infraestructura NIDS (Network Intrusion Detection System) para PIMES," Universidad Politécnica de Valencia, 2017.

[104] A. A. Ghorbani, W. Lu, and M. Tavallaee, "Network intrusion detection and prevention: Concepts and Techniques," *Adv. Inf. Secur.*, 2010.

[105] "OSSEC: Sistema de detección de intrusos," *Mancomún*, 2017. https://www.mancomun.gal/es/artigo-tic/ossec-sistema-de-deteccion-de-intrusos/.

[106] P. V. Cuervo, "¿Qué es Elasticsearch?," *Arquitecto IT*, 2019. http://www.arquitectoit.com/elasticsearch/que-es-elasticsearch/.

[107] "PostgreSQL vs MySQL - ¿Cuál es mejor?," *Guiadev*, 2018. https://guiadev.com/postgresql-vs-mysql/.

[108] M. Uranga, "SPLUNK: Un SIEM para controlarlos a todos," *Jaymon Security*, 2020. https://jaymonsecurity.com/splunk-un-siem-para-controlarlos-a-todos/.

[109] R. Trost, "Part IV: Security and IDS Management," in *Practical Intrusion Analysis: Prevention and Detection for the Twenty-First Century: Prevention and Detection for the Twenty-First Century*, O'Reilly, 2005, p. 163.

[110] E. J. Mira Alfaro, "Implantación de un Sistema de Detección de Intrusos en la Universidad de Valencia," Universidad Politécnica de Valencia, 2001.

[111] Y. Yang, K. Zheng, C. Wu, and Y. Yang, "Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network," *Sensors (Switzerland)*, 2019, doi: 10.3390/s19112528.

[112] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, "Network intrusion detection system: A systematic study of machine learning and deep learning approaches," *Trans. Emerg. Telecommun. Technol.*, 2020, doi: 10.1002/ett.4150.

ANEXO A: HERRAMIENTAS UTILIZADAS EN EL DESARROLLO DE LOS ALGORITMOS

WEKA 150
Spyder 153

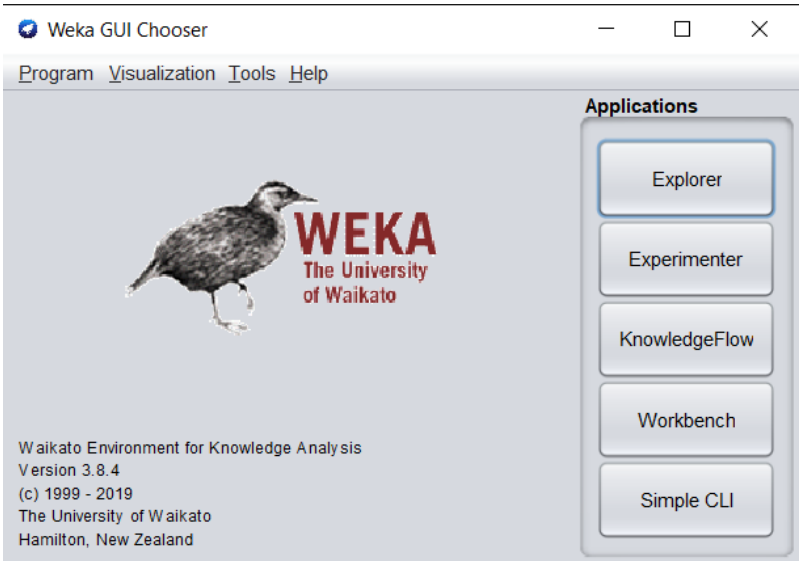
WEKA

WEKA (Waikato Environment for Knowledge Analysis) es una plataforma software dedicada al aprendizaje automático y la minería de datos. Está escrita en Java y, como su nombre indica, ha sido desarrollada en la Universidad de Waikato (Nueva Zelanda).

En primera instancia, WEKA fue desarrollada en un front-end TCL/TK para modelar algoritmos implementados en otros lenguajes de programación, más unas unidades destinadas al preprocesado de datos escritas en C, dedicadas a experimentos de aprendizaje automático. Sin embargo, debido al empuje de Java se decantaron por migrar el software desarrollado a este lenguaje de programación. En el año 2005 esta aplicación recibió el galardón "Data Mining and Knowledge Discovery Service" y un año más tarde, la empresa Pentaho (ahora fusionada junto con Hitachi) adquirió la licencia para usar WEKA para Business Intelligence, naciendo posteriormente Pentaho Business Intelligence, un software pensado para tareas de minería de datos y análisis predictivo [1].

Este software ofrece implementaciones de una gran cantidad de algoritmos de aprendizaje máquina que se pueden aplicar fácilmente al set de datos con el que se esté trabajando. Asimismo, incluye también diversas herramientas para transformar conjuntos de datos, como por ejemplo algoritmos para discretización y muestreo. En definitiva, se puede realizar el proceso completo del desarrollo de un algoritmo de aprendizaje máquina sin tener que programar una sola línea: preprocesar el conjunto de datos que se va a usar y entrenar un algoritmo ML con él, y posteriromente analizar el resultado del clasificador y su rendimiento según las métricas obtenidas.

La aplicación incluye los métodos correspondientes a los problemas principales relativos a la minería de datos: regresión, clasificación, clustering, selección de atributos y asociación de reglas. También se proporcionan numerosas herramientas de visualización y procesamiento de datos, puesto que en el desarrollo de un algoritmo de machine learning resulta de vital importancia tener una idea clara sobre las características del conjunto de datos con el que se trabaja. La interfaz de inicio de WEKA presenta las siguientes opciones [2]:



- *Explorer*[25]*:* Esta funcionalidad es la que permite explotar la gran mayoría de funcionalidades de WEKA mencionadas anteriormente. Por ejemplo, se puede cargar una base de datos y construir un árbol de decisión con estos datos. Además, se facilita mucho la comprensión de cada una de las opciones que se pueden usar, puesto que, si se pasa el ratón por encima de cada botón, aparece un pequeño mensaje mostrando más información acerca de su uso. Los valores predeterminados que utiliza WEKA en sus herramientas aseguran resultados sin tener que realizar un gran esfuerzo. No obstante, es necesario saber qué se está haciendo para poder interpretar correctamente los resultados y poder así cambiar los parámetros de la herramienta que se use.

- *Experimenter:* Esta interfaz está diseñada para automatizar el proceso de ejecución de distintos clasificadores con diferentes parámetros, para así recopilar estadísticas acerca de ellos y poder encontrar el valor óptimo para la técnica de aprendizaje máquina que se esté usando. A través de esta opción, los usuarios más familiarizados con WEKA pueden incluso balancear la carga computacional en múltiples máquinas usando una invocación remota de Java. De esta forma, se pueden poner en marcha experimentos estadísticos a gran escala.

- *KnowledgeFlow:* Permite diseñar configuraciones para el proceso de datos en tiempo real. A diferencia de la interfaz *Explorer*, que mantiene todo en la memoria principal y carga todo el conjunto de datos a la vez, esta interfaz contiene algoritmos incrementales que pueden usarse para el procesamiento de sets de datos muy grandes. Permite arrastrar cajas que representan los algoritmos de aprendizaje máquina y los conjuntos de datos, y unirlos en la disposición que se desee, permitiendo esto la especificación de un flujo de datos completo a través de la conexión de componentes que representan fuentes de datos, herramientas de preprocesado, algoritmos ML, métodos de evaluación y de visualización de los resultados.

[25] *La interfaz que aparece al pulsar esta opción es la mostrada en el Capítulo 2 del presente trabajo, cuando se usaba Weka para el desarrollo de uno de los artículos escogidos.*

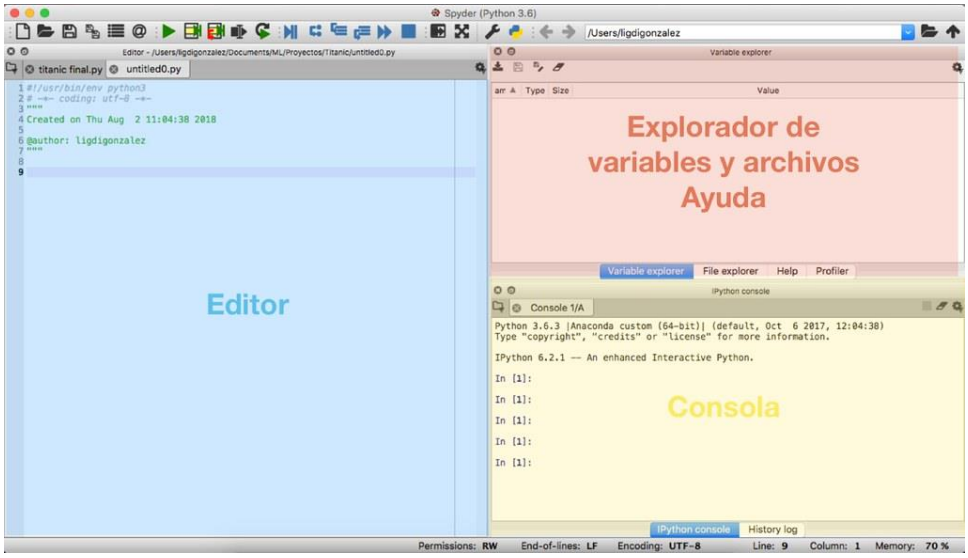
- *Workbench:* En este apartado se combinan las otras tres interfaces (y cualquier plugin que el usuario haya instalado) en un mismo lugar. Es altamente configurable, permitiendo elegir al usuario qué plugins y aplicaciones aparecerán, así como las características de cada uno de ellos.

- *Simple CLI:* Esta opción permite el manejo de WEKA a través de un prompt de comandos.

Cuando se trabaja con este software, un recurso importante es su documentación online, la cual ha sido generada automáticamente a partir del código fuente y refleja claramente la estructura de éste. Esta documentación enumera todos los algoritmos disponibles hasta el día de hoy, ya que WEKA está en constante crecimiento y, al generarse dicha documentación a partir del código fuente, se asegura que esté completamente actualizada. Esta herramienta puede obtenerse en el siguiente enlace: http://www.cs.waikato.ac.nz/ml/weka [2].

Spyder

Spyder es un potente entorno de desarrollo interactivo y multiplataforma para el lenguaje de programación Python. Posee funciones avanzadas de edición, pruebas interactivas, etc. Gracias al soporte de IPython (intérprete interactivo mejorado de Python), al amplio catálogo de paquetes que ofrece este lenguaje de programación y a que es software libre, se ha convertido en el IDE más extendido para Python. Además, incluye soporte de herramientas interactivas para la inspección de datos e incorpora controles de calidad específicos de Python e instrumentos como Pyflakes o Rope. La siguiente imagen muestra el entorno de trabajo de Spyder [3]:



Sus características generales son las que se enumeran a continuación [4]:

- El editor que integra es multilenguaje. Posee un navegador de función, funciones de análisis de código, opción de finalización de código, etc.

- La consola es interactiva. La integración de la consola IPython en este IDE permite evaluar al momento el código que se está desarrollando por línea, bloque o al completo, así como renderizar los gráficos justo después de la salida del código o en ventanas interactivas.

- El visor de documentación que tiene permite la consulta de cualquier documento para cualquier llamada de clase o función ejecutada en el editor.
- Se pueden explorar las variables con las que se está trabajando, así como editarlas con un editor basado en GUI que posee Spyder.

- Las herramientas de desarrollo que ofrece son muy variadas: se puede analizar el código mediante un analizador estático, así como depurarlo con el depurador. También se aceptan expresiones regulares a la hora de utilizar el buscador.

Como ya se mencionó anteriormente, la biblioteca de librerías que posee Python es muy extensa y variada, y para el desarrollo de la parte práctica del presente Trabajo se ha hecho uso de algunas de ellas, las cuales se comentan brevemente a continuación:

- **Numpy**: es una biblioteca que da soporte a la creación de grandes vectores y matrices multidimensionales, junto con una gran biblioteca de funciones matemáticas de alto nivel para poder operar con ellas. Es de código abierto y cuenta con una gran comunidad detrás apoyándola [5].

- **Pytorch**: esta librería está diseñada para la ejecución de cálculos numéricos con tensores. Además, permite la utilización de la GPU del equipo (si dispone de ella) para acelerar la velocidad de los cálculos. A diferencia de otros paquetes que realizan las mismas tareas como TensorFlow, Pytorch trabaja con grafos dinámicos en lugar de estáticos, lo que implica que en tiempo de ejecución se pueden modificar las funciones y el cálculo del gradiente cambiará con estos cambios [6].

- **Scikit-learn**: contiene muchos algoritmos de clasificación, regresión y análisis de agrupaciones, entre los cuales se cuentan SVM, árboles de decisión, K-means, DBSCAN o *gradient boosting* [7].

- **Matplotlib**: es una biblioteca dedicada a la creación de gráficas a partir de datos contenidos en listas o vectores NumPy. Proporciona una API, pylab, diseñada para emular a la de MATLAB [8].

- **Pandas**: es un paquete que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para facilitar el trabajo con datos estructurados [9].

BIBLIOGRAFÍA DE LOS ANEXOS

[1]. España, R. (2020, 9 julio). ¿Qué es Weka y qué tiene que ver con Big Data? B12. https://agenciab12.com/noticia/que-es-weka-que-tiene-que-ver-bigdata?utm_source=social&utm_medium=facebook_es

[2]. The Weka Workbench (2016). Eibe Frank, Mark A. Hall, Ian H. Witten.

[3]. A., D. (2017, 8 diciembre). Spyder, un potente entorno de desarrollo interactivo para Python. Ubunlog. https://ubunlog.com/spyder-entorno-desarrollo-python/

[4]. *spyder*. (2020, 8 noviembre). PyPI. https://pypi.org/project/spyder/

[5]. Wikipedia – Numpy. https://es.wikipedia.org/wiki/NumPy

[6]. Vieco, J. (2018, 9 enero). Pytorch: ¿Qué es y como se... Cleverpy Machine Learning. https://cleverpy.com/que-es-pytorch-y-como-se-instala/

[7]. colaboradores de Wikipedia. (2020b, noviembre 8). Scikit-learn. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Scikit-learn

[8]. colaboradores de Wikipedia. (2020b, abril 27). Matplotlib. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Matplotlib

[9]. pandas. (2020, 7 diciembre). PyPI. https://pypi.org/project/pandas/

ANEXO B: CÓDIGOS USADOS EN EL TRABAJO

visualizacionsets.py 158
metricas.py 163
articulo_I.py 164
articulo_II.py 168
preprocessing.py 172
model.py 175
cvae.py 178
valuate.py 183
mlp.py 186

```
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

visualizacion_sets.py
Este código fusiona los 4 CSV's que conforman el conjunto completo de datos de
UNSW-NB15 en uno, así como el training set y test set del conjunto reducido.
Corrige también errores en el etiquetado de los ataques y realiza un análisis
visual de la estructura del conjunto de datos, tanto en su versión
completa como en la reducida.
"""


# Imports de librerías:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.compose import ColumnTransformer
sns.set()


# ============================================================================
#  AGRUPACIÓN DE TODO EL CONJUNTO DE DATOS EN 1 CSV
# ============================================================================

#Unimos todos los csv en uno solo.
dataset_1 = pd.read_csv('UNSW-NB15_1.csv', header=None)
dataset_2 = pd.read_csv('UNSW-NB15_2.csv', header=None)
dataset_3 = pd.read_csv('UNSW-NB15_3.csv', header=None)
dataset_4 = pd.read_csv('UNSW-NB15_4.csv', header=None)
dataset=pd.concat([dataset_1,dataset_2,dataset_3,dataset_4], axis=0)

# Unimos trainset y testset en un mismo csv para visualizar los datos.
trainset = pd.read_csv('UNSW_NB15_training-set.csv',header=0)
testset = pd.read_csv('UNSW_NB15_testing-set.csv',header=0)
dataset_red = pd.concat([trainset,testset], axis=0)


# #Etiquetamos las columnas de los archivos .CSV:
dataset.columns = ['srcip','sport','dstip','dsport','proto','state','dur',
                   'sbytes','dbytes','sttl','dttl','sloss','dloss', 'service',
                   'sload','dload','spkts','dpkts','swin','dwin','stcpb',
                   'dtcpb','smean','dmean','trans_depth','res_bdy_len',
                   'sjit', 'djit','stime','ltime','sintpkt','dintpkt','tcprtt',
                   'synack','ackdat','is_sm_ips_ports','ct_state_ttl',
                   'ct_flw_http_mthd','is_ftp_login','ct_ftp_cmd','ct_srv_src',
                   'ct_srv_dst','ct_dst_ltm','ct_src_ltm','ct_src_dport_ltm',
                   'ct_dst_sport_ltm','ct_dst_src_ltm','attack_cat','label']


dataset_red.columns = ['id','dur','proto','service','state','spkts','dpkts',
                   'sbytes','dbytes','rate','sttl','dttl','sload','dload',
                   'sloss','dloss','sintpkt','dintpkt','sjit', 'djit',
                   'swin','stcpb','dtcpb','dwin','tcprtt','synack','ackdat',
                   'smean','dmean','trans_depth','res_bdy_len', 'ct_srv_dst',
                   'ct_state_ttl','ct_dst_ltm','ct_src_dport_ltm','ct_dst_sport_ltm',
                   'ct_dst_src_ltm','is_ftp_login','ct_ftp_cmd','ct_flw_http_mthd',
                   'ct_src_ltm','ct_srv_src','is_sm_ips_ports','attack_cat','label']


#eliminamos los espacios para evitar duplicados:
dataset['attack_cat']=dataset['attack_cat'].str.replace(' Fuzzers ','Fuzzers')
dataset['attack_cat']=dataset['attack_cat'].str.replace(' Fuzzers','Fuzzers')
```

```
dataset['attack_cat']=dataset['attack_cat'].str.replace(' Reconnaissance ','Reconnaissance')
dataset['attack_cat']=dataset['attack_cat'].str.replace(' Shellcode ','Shellcode')
dataset['attack_cat']=dataset['attack_cat'].str.replace('Backdoors','Backdoor')

#cambiamos con esta línea los NaN del tráfico normal por Normal.
#Seleccionamos la columna attack_cat y cambiamos los nan por normal.
dataset['attack_cat'].fillna('Normal', inplace=True)

# Eliminamos estas características con las que no se trabaja generalmente.
dataset= dataset.drop(['srcip','sport','dstip','dsport'],1)
dataset_red=dataset_red.drop(['id',],1)


# La categoría de ct_ftp_cmd es object, y no int64. La convertimos.
dataset['ct_ftp_cmd']=pd.to_numeric(dataset['ct_ftp_cmd'], errors='coerce').fillna(0).astype('int64')
dataset=dataset.fillna(0)

# sobreescribimos el .CSV de nuevo con los cambios realizados. Set completo
dataset.to_csv('UNSW-NB15_full_label.csv', index=None)

# Unimos en un csv el set de datos reducido (estaba separado en train y test).
dataset_red.to_csv('UNSW-NB15_red.csv', index=None)

#%%
# ================================================================================
#                   ANÁLISIS VISUAL DEL CONJUNTO DE DATOS
# ================================================================================

pd.set_option('precision', 3, 'display.max_columns', None) #cuántos decimales se van a poner en la precisión

# Preguntamos de qué set queremos extraer las características:

print("Para visualizar el conjunto de datos completo del UNSW-NB15, inserte 1.\n")
      print("Para visualizar el conjunto de datos REDUCIDO del UNSW-NB15, inserte 2.\n")
      datos = input("Inserte un número: ")
      print("\n")

      while datos != '1' and datos != '2':
          print("VALOR NO RECONOCIDO.\n")
          print("Para visualizar el conjunto de datos completo del UNSW-NB15, inserte 1.\n")
          print("Para visualizar el conjunto de datos REDUCIDO del UNSW-NB15, inserte 2.\n")
          datos = input("Inserte un número: ")
          print("\n")

      if datos == '1':
          print("VISUALIZACIÓN DE CONJUNTO DE DATOS COMPLETO.\n")
      #con header=0 hacemos que la 1º fila sea el nombre de las columnas
          dataset = pd.read_csv('UNSW-NB15_full_label.csv', header=0)
          value="set completo"
          offset=10000   #variable para subir los números de los ataques y despegarlos de la columna

      else:
          print("VISUALIZACIÓN DE CONJUNTO DE DATOS REDUCIDO.\n")
          dataset = pd.read_csv('UNSW-NB15_red.csv', header=0)
          value="set reducido"
          offset=1000 #variable para subir los números de los ataques y despegarlos de la columna


      # Eliminamos la característica binaria label para la visualización
```

```
126.        dataset= dataset.drop(['label'],1)
127.
128.        # ================================================================================
129.        #                          ESTADÍSTICAS DEL DATASET
130.        # ================================================================================
131.
132.                                ###### ESTADÍSTICAS BÁSICAS ######
133.
134.        print("Tamaño del set de datos: \n")
135.        print(dataset.shape) #filas y columnas de los datos
136.
137.        print("Tipo de dato de las características: \n")
138.        print(dataset.dtypes) #tipo de dato de cada columna
139.
140.        print(dataset.head(10))  #muestra las 10 primeras filas del archivo
141.
142.
143.        print("Estadísticas del {}: \n".format(value))
144.        print(dataset.describe()) #genera estadísticas de los datos.
145.
146.
147.
148.                            ###### CANTIDAD DE ATAQUES DEL DATASET ######
149.
150.
151.        # Número de ataques existentes en el dataset completo:
152.        print('\n')
153.        print("Número de instancias de cada categoría en el {}: \n".format(value))
154.        ataques = dataset['attack_cat'].value_counts()
155.        print(ataques)
156.        print('\n')
157.
158.
159.        nom_atk =["Normal","Generic","Exploits","Fuzzers","DoS","Reconnaissance","Analys
    is",
160.                  "Backdoor","Shellcode","Worms"]
161.
162.        plt.close('all')
163.
164.        #Cambiamos tamaño de la figura
165.        plt.figure(figsize=(9,6))
166.
167.        # Ajustamos tamaño de los ejes X e Y.
168.        y_pos = [0,2,4,6,8,10,12,14,16,18]
169.        x_pos = [-0.5,1.5,3.5,5.5,8,8.75,11.5,13.5,15.5,17.5]
170.
171.        x_num = [-0.7,1.3,3.3,5.3,7.4,9.4,11.5,13.5,15.5,17.5]
172.
173.        # Colores de cada barra:
174.        colores = ['#1f98b4', '#ff7f0e', '#2ca02c', '#d62748', '#9497bc',
175.                   '#8c557b', '#da17c2', '#7f7f7f', '#bcbd22', '#17becf']
176.
177.        # Diagrama de barras:
178.        plt.bar(y_pos, ataques, width=1.2, color=colores)
179.        plt.title('Registro de ataques en el {}'.format(value), fontsize=14, fontweight=
    'bold')
180.
181.
182.        # Ponemos la cantidad de elementos de cada clase encima de la barra
183.        for i in range(len(x_pos)):
184.            plt.text(x = x_num[i] , y = ataques[i]+offset, s = ataques[i], size = 12)
185.
186.        # Ajustamos los nombres de los ataques en el eje X.
187.        plt.xticks(x_pos, nom_atk, rotation=45)
188.
```

```
189.       # Metemos un poco de relleno en el eje Y para que los números de las columnas no
190.       # queden pegados al borde de la cuadrícula
191.       plt.margins(y=0.12)
192.
193.       plt.show()
194.
195.
196.                          ###### CORRELACIÓN DE CARACTERÍSTICAS ######
197.
198.       plt.figure(figsize=(12,10))
199.       plt.title('Matriz de correlación en el {}'.format(value), fontsize=14, fontweigh
    t='bold')
200.       correl = dataset.corr()
201.       sns.heatmap(correl)
202.       plt.show()
203.
204.
205.                                  ###### T-SNE SCATTER PLOT ######
206.
207.       # Normalización y Label Encoding para aplicar t-SNE.
208.
209.       # Identificamos las features categóricas para poder convertirlas a enteros:
210.       for col_name in dataset.columns:
211.           if dataset[col_name].dtypes == 'object' :
212.               le = preprocessing.LabelEncoder()
213.               unique_cat = len(dataset[col_name].unique())
214.               print("El feature '{col_name}' tiene {unique_cat} categorías".format(col
    _name=col_name, unique_cat=unique_cat))
215.               dataset[col_name] = le.fit_transform(dataset[col_name])
216.               mapeo = dict(zip(le.classes_, le.transform(le.classes_)))
217.               print(mapeo)
218.               print()
219.
220.
221.       # En numcol guardamos las variables numéricas para aplicar la normalización está
    ndar
222.       numcol=dataset.columns
223.       numcol=numcol.drop(['proto','service','state','attack_cat'])
224.       numcol=pd.Index(numcol)
225.       numcol=numcol.to_list()
226.
227.
228.       # guardamos el nombre de las columnas y el tipo de datos para despues convertir
    a df
229.       listacols=numcol + ['proto','service','state','attack_cat']
230.       types=dataset.dtypes
231.
232.       # reordenamos los tipos de datos para asociarlos a las features adecuadas
233.       types= types.reindex(listacols)
234.
235.
236.       # Normalizamos las variables numéricas con ColumnTransformer
237.       sca=StandardScaler()
238.       column_trans = ColumnTransformer([('scaler', sca, numcol)], remainder='passthrou
    gh')
239.       dataset= column_trans.fit_transform(dataset)
240.
241.
242.       # reconvertimos a DataFrame y volvemos a poner los tipos de datos (con ColumnTra
    nsformer
243.       # se vuelven todos object)
244.       dataset=pd.DataFrame(dataset,columns=listacols)
245.       retype = dict(zip(listacols, types.tolist()))
246.       dataset = dataset.astype(retype)
247.
248.
```

```
249.        print()
250.
251.
252.        # cogemos num_samples muestras de cada una de las clases para efectuar el t-
    SNE y
253.        # que no haya problemas de memoria.
254.        num_samples=100
255.
256.        nor = dataset.query('attack_cat == 6').sample(num_samples)
257.        an = dataset.query('attack_cat == 0').sample(num_samples)
258.        back = dataset.query('attack_cat == 1').sample(num_samples)
259.        dos = dataset.query('attack_cat == 2').sample(num_samples)
260.        expl = dataset.query('attack_cat == 3').sample(num_samples)
261.        fuzz = dataset.query('attack_cat == 4').sample(num_samples)
262.        gen = dataset.query('attack_cat == 5').sample(num_samples)
263.        reco = dataset.query('attack_cat == 7').sample(num_samples)
264.        shell = dataset.query('attack_cat == 8').sample(num_samples)
265.        worm = dataset.query('attack_cat == 9').sample(num_samples)
266.
267.        # Concatenamos las 100 muestras aleatorias de cada clase que hemos escogido en
268.        # un vector llamado setred
269.        setred=pd.concat([nor,an,back,dos,expl,fuzz,gen,reco,shell,worm],axis=0)
270.
271.        tsne = TSNE(n_components=2, verbose=1, perplexity=40, n_iter=300).fit_transform(
    setred)
272.
273.
274.        # hay 40 muestras de cada clase. La última muestra de la primera clase está en

275.        # la posición 39, la ultima de la 2da clase en el fin_index+40, etc.
276.        fin_index=num_samples-1
277.
278.        fig = plt.figure(figsize=(13,13))
279.        ax1 = fig.add_subplot(111)
280.
281.
282.        ax1.scatter(tsne[0:fin_index, 0], tsne[0:fin_index, 1], 20, colores[0],label='No
    rmal')
283.        ax1.scatter(tsne[fin_index+1:fin_index*2, 0], tsne[fin_index+1:fin_index*2, 1],
    20, colores[6],label='Analysis')
284.        ax1.scatter(tsne[(fin_index*2)+1:fin_index*3, 0], tsne[(fin_index*2)+1:fin_index
    *3, 1], 20, colores[7],label='Backdoor')
285.        ax1.scatter(tsne[(fin_index*3)+1:fin_index*4, 0], tsne[(fin_index*3)+1:fin_index
    *4, 1], 20, colores[4],label='DoS')
286.        ax1.scatter(tsne[(fin_index*4)+1:fin_index*5, 0], tsne[(fin_index*4)+1:fin_index
    *5, 1], 20, colores[2],label='Exploits')
287.        ax1.scatter(tsne[(fin_index*5)+1:fin_index*6, 0], tsne[(fin_index*5)+1:fin_index
    *6, 1], 20, colores[3],label='Fuzzers')
288.        ax1.scatter(tsne[(fin_index*6)+1:fin_index*7, 0], tsne[(fin_index*6)+1:fin_index
    *7, 1], 20, colores[1],label='Generic')
289.        ax1.scatter(tsne[(fin_index*7)+1:fin_index*8, 0], tsne[(fin_index*7)+1:fin_index
    *8, 1], 20, colores[5],label='Reconnaissance')
290.        ax1.scatter(tsne[(fin_index*8)+1:fin_index*9, 0], tsne[(fin_index*8)+1:fin_index
    *9, 1], 20, colores[8],label='Shellcode')
291.        ax1.scatter(tsne[(fin_index*9)+1:fin_index*10, 0], tsne[(fin_index*9)+1:fin_inde
    x*10, 1], 20, colores[9],label='Worms')
292.
293.        plt.title('Scatter de un subconjunto del UNSW-NB15 después de aplicar t-
    SNE para el {} '.format(value), fontsize=14, fontweight='bold')
294.        plt.legend(loc='lower left')
295.        plt.show()
```

```python
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

metricas.py

Código encargado de sacar las métricas correspondientes a los resultados
obtenidos por el clasificador. La función performance es llamada desde los
archivos articulo_I.py y articulo_II.py.
"""


# Imports de librerías:
from sklearn.metrics import (precision_score, recall_score,f1_score, confusion_matrix,
accuracy_score,classification_report)
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns


# ============================================================================
#  Función performance: Se encarga de calcular las métricas para el clasificador,
#                       así como de trazar la matriz de confusión.
#                       las métricas calculadas son F1-Score, accuracy, recall
#                       y precision.
#       Entradas: datos_esp: etiquetas verdaderas para cada instancia de test
#                 datos_pred: etiquetas predecidas por el clasificador para
#                             cada instancia de test
#                 labels: nombres de las clases
# ============================================================================

def performance(datos_esp, datos_pred, labels):
    pl_matconf(datos_esp, datos_pred, labels)
    print("Estadísticas por categoría y generales: \n")
    print(classification_report(datos_esp, datos_pred,labels,digits=4))


# ============================================================================
#  Función pl_matconf: Se encarga de trazar la matriz de confusión para la
#                      clasificación realizada por el clasificador, dando el
#                      formato adecuado al gráfico.
#       Entradas: datatrue: etiquetas verdaderas para cada instancia de test
#                 datapred: etiquetas predecidas por el clasificador para
#                             cada instancia de test
#                 label_list: nombres de las clases
# ============================================================================
def pl_matconf(datatrue, datapred, label_list):

    # Cálculo de la matriz de confusión
    cm = confusion_matrix(datatrue, datapred, label_list)

    fig, ax = plt.subplots(figsize = (10, 10))

    sns.set(font_scale=1.5) #aumentamos tamaño letra de la matriz de confusión

    # Annot=true anota los números en cada casilla. cmap sirve para elegir los
    # colores de la matriz de confusión, mientras que fmt sirve para determinar
    # el formato de los números de cada casilla.
    sns.heatmap(cm, annot=True, ax = ax, cmap='PuBu',fmt='d',
                xticklabels=label_list, yticklabels=label_list);

# Título del gráfico y etiquetas de los ejes X e Y
    ax.set_xlabel('Predicción');ax.set_ylabel('Esperado');
    ax.set_title('Matriz de confusión');
```

```
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

articulo_I.py

Realiza el preprocesado y posterior clasificación binaria de la base de datos
UNSW-NB15 con un clasificador Random Forest, siguiendo las directrices del
artículo "Effects of Feature Selection and Normalization on Network
Intrusion Detection", de Mubarak Albarka Umar y Chen Zhanfang.
También se realiza un ajuste de hiperparámetros al final del código con la
intención de mejorar los parámetros considerados en primera instancia en el RF.
(En principio se deja comentado el ajuste al no encontrar mejoría en los
resultados).
"""

# Imports de librerías:
import pandas as pd
from metricas import *
from sklearn.preprocessing import MinMaxScaler
from time import time
from sklearn.model_selection import GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier

# ============================================================================
#           DIVISIÓN DE DATOS EN SETS DE TRAIN Y TEST
# ============================================================================

# La base de datos utilizada es la UNSW-NB15, creada por Nour Moustafa en 2015.
# Este código se desarrolla con el conjunto de entrenamiento de la versión reducida
# de la UNSW-NB15, previa selección de características, división en test y
# entrenamiento y filtrado de datos (esto se realiza en WEKA).
# De aquí salen training_bin_after_fs y testing_bin_after_fs.
# La bbdd se puede encontrar en https://www.unsw.adfa.edu.au/unsw-canberra-
cyber/cybersecurity/ADFA-NB15-Datasets/
# Está compuesta por nueve tipos de ataques en total: Fuzzers, Analysis, Backdoors, DoS
, Exploits,
# Generic, Reconnaissance, Shellcode and Worms.

#Con header=0 hacemos que la 1º fila sea el nombre de las columnas.
train = pd.read_csv('training_bin_after_fs.csv', header=0)
test = pd.read_csv('testing_bin_after_fs.csv', header=0)

# Se verifica que no hay valores nulos en el dataset.
print(train.isnull().values.any())
print(test.isnull().values.any())

# Separamos la característica label del resto de características.
# Lo hacemos para el conjunto de test y para el de training.
X_train = train.drop(['label'],1)
Y_train = train['label']

X_test = test.drop(['label'],1)
Y_test = test['label']


label_bin = Y_test.unique()

# Variables categóricas del conjunto de datos:
features = ['proto','service','state']

# ============================================================================
#                           NORMALIZACIÓN
# ============================================================================

# Eliminamos las vbles categóricas de las columnas para poder quedarnos con las
```

```
66. # vbles numéricas (a las que aplicaremos la normalización)
67. numcol = X_train.columns
68. numcol = numcol.drop(features)
69. numcol = pd.Index(numcol)
70. numcol = numcol.to_list()
71. 
72. # guardamos el nombre de las columnas y el tipo de datos para despues convertir a DataF
    rame
73. # y seguir procesando los datos de esta forma.
74. listacols = numcol + features #añadimos las vbles categóricas al final
75. types = X_train.dtypes
76. 
77. 
78. # reordenamos los tipos de datos para asociarlos a las features adecuadas
79. types = types.reindex(listacols)
80. 
81. # normalizamos solo las categorías numéricas:
82. sca = MinMaxScaler()
83. 
84. # con passthrough hacemos que el resto de columnas las deje como está
85. column_trans = ColumnTransformer([('scaler', sca, numcol)],remainder='passthrough')
86. 
87. X_test = column_trans.fit_transform(X_test)
88. 
89. X_train = column_trans.fit_transform(X_train)
90. 
91. # reconvertimos a DataFrame
92. X_train=pd.DataFrame(X_train,columns=listacols)
93. X_test=pd.DataFrame(X_test,columns=listacols)
94. 
95. 
96.         # ================================================================================
97. #             DIFERENCIA DE ETIQUETAS EN SETS DE TRAIN Y TEST
98. # ================================================================================
99. 
100.        # Completamos con columnas vacías las categorías faltantes en ambos conjuntos.
101. 
102.        def labels(tra,tes,feat):
103.            trainset=tra[feat].tolist()
104.            testset= tes[feat].tolist()
105.            diff1=list(set(trainset) - set(testset))
106.            diff2=list(set(testset) - set(trainset))
107.            return(diff1, diff2)
108. 
109. 
110.        for feat in features:
111. 
112.            dif1,dif2 = labels(X_train, X_test, feat)
113.            print('FEATURE ' + feat + '. Faltan en el set de testing: ')
114.            print(dif1)
115.            if len(dif1) != 0:
116.                print('Completamos el set de test con las categorías que faltan...')
117.                for ind in range(len(dif1)):
118.                    X_test.insert(len(X_test.columns),dif1[ind],0)
119. 
120. 
121.            print('FEATURE ' + feat + '. Faltan en el set de entreno: ')
122.            print(dif2)
123.            if len(dif2) != 0:
124.                print('Completamos el set de entreno con las categorías que faltan...')
125.                for ind in range(len(dif2)):
126.                    X_train.insert(len(X_train.columns),dif2[ind],0)
127.                print("\n")
128. 
129.        print("\n")
130. 
```

```
131.        # ================================================================================

132.        #                              ONE-HOT ENCODING
133.        # ================================================================================

134.
135.        # Creamos una función para aplicar a todas las variables categóricas:
136.        def onehot(dataframe_orig, feat_a_codif):
137.            dummies = pd.get_dummies(dataframe_orig[feat_a_codif])
138.            #añadimos al array original la codificación one-hot
139.            res = pd.concat([dataframe_orig, dummies], axis=1)
140.            #tiramos la columna de la feature convertida
141.            res = res.drop([feat_a_codif], axis=1)
142.            return(res)
143.
144.
145.        # Identificamos las features categóricas (tipo object) para poder convertirlas a
    enteros:
146.        print('Train set:')
147.        for col_name in X_train.columns:
148.            if col_name == 'proto' or col_name == 'service' or col_name == 'state' :
149.                unique_cat = len(X_train[col_name].unique())
150.                print("El feature '{col_name}' tiene {unique_cat} categorías".format(col
    _name=col_name, unique_cat=unique_cat))
151.                X_train=onehot(X_train, col_name)
152.
153.        print("\n")
154.
155.        print('Test set:')
156.        for col_name in X_test.columns:
157.            if col_name == 'proto' or col_name == 'service' or col_name == 'state' :
158.                unique_cat = len(X_test[col_name].unique())
159.                print("El feature '{col_name}' tiene {unique_cat} categorías".format(col
    _name=col_name, unique_cat=unique_cat))
160.                X_test=onehot(X_test, col_name)
161.
162.        print("\n")
163.
164.
165.        # ================================================================================

166.        #                         CLASIFICACIÓN CON ALGORITMOS
167.        # ================================================================================

168.
169.        # Parámetros por defecto del random forest:
170.        # max_depth = None
171.        # min_samples_leaf = 1
172.        # max_samples_split = 2
173.        # max_features = 'auto'
174.
175.        # Entrenamos el random forest
176.        tree_clf = RandomForestClassifier(n_estimators=400, n_jobs=3, verbose=1)
177.
178.        tiempo_inicial = time()
179.        tree_clf.fit(X_train, Y_train)
180.
181.        # Realizamos la clasificación con el set de test
182.        Y_pred = tree_clf.predict(X_test)
183.        tiempo_final = time()
184.        tiempo_ejecucion = tiempo_final - tiempo_inicial
185.
186.        print('El tiempo de ejecución fue de: %.3f segundos' % tiempo_ejecucion)
187.
188.        # Sacamos las métricas del clasificador:
189.        performance(Y_test,Y_pred,label_bin)
190.
```

```
191.
192.        # ================================================================================

193.        #                          HYPERPARAMETER TUNING
194.        # ================================================================================

195.        #
196.
197.        # Creamos la rejilla de parámetros
198.        # param_grid = {
199.        #     'max_depth': [100, 110, None],
200.        #     'max_features': [10, 15, 'auto'],
201.        #     'min_samples_leaf': [1,40,50],
202.        #     'min_samples_split': [2,30,50],
203.        #     'n_estimators': [300, 400, 1000]
204.        # }
205.
206.        # # Creación del modelo
207.        # rf = RandomForestClassifier()
208.
209.        # # Instanciamos la búsqueda de rejilla:
210.        # grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
211.        #                            cv = 4, n_jobs = -
    1, verbose = 2, scoring='f1_macro')
212.
213.        # grid_search.fit(X_train, Y_train)
214.
215.        # print('La combinación de parámetros que mejor métrica objetivo arroja es: ')
216.        # print(grid_search.best_params_)
217.        # print('El valor del F1-
    Score con la mejor combinación de parámetros es: ', grid_search.best_score_)
```

```
# -*- coding: utf-8 -*-

"""
@author: María Dolores Trapero Estepa

articulo_II.py

Realiza el preprocesado y posterior clasificación binaria y multiclase de la
base de datos UNSW-NB15 con un clasificador Random Forest, siguiendo las
directrices del artículo "Evaluation of Machine Learning Algorithms for
Anomaly Detection", de Nebrase Elmbrabit, Feixiang Zhou, Fengyin Li y Huiyu Zhou.
También se realiza un ajuste de hiperparámetros al final del código con la
intención de mejorar los parámetros considerados en primera instancia en el RF.
"""


# Imports de librerías:
import pandas as pd
from metricas import *
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from time import time
from sklearn.model_selection import GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier


# ============================================================================
#           DIVISIÓN DE DATOS EN SETS DE TRAIN Y TEST
# ============================================================================

#con header=0 hacemos que la 1º fila sea el nombre de las columnas
dataset = pd.read_csv('UNSW-NB15_full_label.csv', header=0)


# Se verifica que no hay valores nulos en el dataset.
print(dataset.isnull().values.any())

# Elegimos qué tipo de clasificación queremos realizar:
print("Para realizar clasificación multiclase, inserte 0.\n")
print("Para realizar clasificación binaria, inserte 1.\n")

datos = input("Inserte un número: ")
print("\n")

while datos != '0' and datos != '1':
    print("VALOR NO RECONOCIDO.\n")
    print("Para realizar clasificación multiclase, inserte 0.\n")
    print("Para realizar clasificación binaria, inserte 1.\n")
    datos = input("Inserte un número: ")
    print("\n")

if datos == '1':
    print("CLASIFICACIÓN BINARIA.\n")
    # Separamos la característica label del resto de características.
    # Lo hacemos para el conjunto de test y para el de training.
    y = dataset['label']
    dataset = dataset.drop(['label','attack_cat'],1)

    X_train, X_test, Y_train, Y_test = train_test_split(dataset, y,
                                                        stratify=y,
                                                        test_size=0.30, random_state=42)


else:
    print("CLASIFICACIÓN MULTICLASE.\n")
    # Separamos la característica attack_cat del resto de características.
```

```
    # Lo hacemos para el conjunto de test y para el de training.
    y = dataset['attack_cat']
    dataset = dataset.drop(['label','attack_cat'],1)

    X_train, X_test, Y_train, Y_test = train_test_split(dataset, y,
                                                        stratify=y,
                                                        test_size=0.30, random_state=42)


# Distribución de variables categóricas en train y test.
features=['proto','service','state']


# ================================================================================
#                            NORMALIZACIÓN
# ================================================================================

# Eliminamos las vbles categóricas de las columnas para poder quedarnos con las
# vbles numéricas (a las que aplicaremos la normalización)
numcol=X_train.columns
numcol=numcol.drop(features)
numcol=pd.Index(numcol)
numcol=numcol.to_list()

# guardamos el nombre de las columnas y el tipo de datos para despues convertir a DataFrame
# y seguir procesando los datos de esta forma.
listacols=numcol + features
types=X_train.dtypes


      # reordenamos los tipos de datos para asociarlos a las features adecuadas
      types= types.reindex(listacols)

      # normalizamos solo las categorías numéricas:
      sca=StandardScaler()

      # con passthrough hacemos que el resto de columnas las deje como está
      column_trans = ColumnTransformer([('scaler', sca, numcol)],remainder='passthrough')

      X_test= column_trans.fit_transform(X_test)

      X_train = column_trans.fit_transform(X_train)

      # reconvertimos a DataFrame
      X_train=pd.DataFrame(X_train,columns=listacols)
      X_test=pd.DataFrame(X_test,columns=listacols)


      # ================================================================================

      #             DIFERENCIA DE ETIQUETAS EN SETS DE TRAIN Y TEST
      # ================================================================================


      # Completamos con columnas vacías las categorías faltantes en ambos conjuntos.

      def labels(tra,tes,feat):
          trainset=tra[feat].tolist()
          testset= tes[feat].tolist()
          diff1=list(set(trainset) - set(testset))
          diff2=list(set(testset) - set(trainset))
          return(diff1, diff2)

```

```
131.
132.       for feat in features:
133.
134.           dif1,dif2 = labels(X_train, X_test, feat)
135.           print('FEATURE ' + feat + '. Faltan en el set de testing: ')
136.           print(dif1)
137.           if len(dif1) != 0:
138.               print('Completamos el set de test con las categorías que faltan...')
139.               for ind in range(len(dif1)):
140.                   X_test.insert(len(X_test.columns),dif1[ind],0)
141.
142.
143.           print('FEATURE ' + feat + '. Faltan en el set de entreno: ')
144.           print(dif2)
145.           if len(dif2) != 0:
146.               print('Completamos el set de entreno con las categorías que faltan...')
147.               for ind in range(len(dif2)):
148.                   X_train.insert(len(X_train.columns),dif2[ind],0)
149.               print("\n")
150.
151.       print("\n")
152.
153.
154.
155.       # ================================================================================
156.       #                              ONE-HOT ENCODING
157.       # ================================================================================
158.
159.       # Creamos una función para aplicar a todas las variables categóricas:
160.       def onehot(dataframe_orig, feat_a_codif):
161.           dummies = pd.get_dummies(dataframe_orig[feat_a_codif])
162.           #añadimos al array original la codificación
163.           res = pd.concat([dataframe_orig, dummies], axis=1)
164.           #tiramos la columna de la feature convertida
165.           res = res.drop([feat_a_codif], axis=1)
166.           return(res)
167.
168.
169.       # Identificamos las features categóricas para poder convertirlas a enteros:
170.       print('Train set:')
171.       for col_name in X_train.columns:
172.           if col_name == 'proto' or col_name == 'service' or col_name == 'state' :
173.               unique_cat = len(X_train[col_name].unique())
174.               print("El feature '{col_name}' tiene {unique_cat} categorías".format(col
    _name=col_name, unique_cat=unique_cat))
175.               X_train=onehot(X_train, col_name)
176.
177.       print("\n")
178.
179.       print('Test set')
180.       for col_name in X_test.columns:
181.           if col_name == 'proto' or col_name == 'service' or col_name == 'state' :
182.               unique_cat = len(X_test[col_name].unique())
183.               print("El feature '{col_name}' tiene {unique_cat} categorías".format(col
    _name=col_name, unique_cat=unique_cat))
184.               X_test=onehot(X_test, col_name)
185.
186.       print("\n")
187.
188.
189.       # ================================================================================
190.       #                        CLASIFICACIÓN CON ALGORITMOS
```

```
191.        # ================================================================================

192.
193.        # Etiquetas para matriz de confusión
194.        labels = Y_test.unique()
195.
196.        # Parámetros por defecto:
197.        # max_depth = None
198.        # min_samples_leaf = 1
199.        # max_samples_split = 2
200.        # max_features = 'auto'
201.
202.        # Entrenamos el random forest
203.        tree_clf = RandomForestClassifier(n_estimators=400, n_jobs=3,verbose=1)
204.
205.        tiempo_inicial = time()
206.        tree_clf.fit(X_train, Y_train)
207.
208.        # Realizamos la clasificación con el set de test
209.        Y_pred = tree_clf.predict(X_test)
210.        tiempo_final = time()
211.        tiempo_ejecucion = tiempo_final - tiempo_inicial
212.
213.        print('El tiempo de ejecución fue: %0.3f segundos.' % tiempo_ejecucion)
214.
215.        # Sacamos las métricas del clasificador:
216.        performance(Y_test,Y_pred,labels)
217.
218.
219.        # ================================================================================

220.        #                         HYPERPARAMETER TUNING
221.        # ================================================================================

222.        #
223.
224.        # # Creamos la rejilla de parámetros
225.        # param_grid = {
226.        #     'max_depth': [100, 110, None],
227.        #     'max_features': [10, 15, 'auto'],
228.        #     'min_samples_leaf': [1,40,50],
229.        #     'min_samples_split': [2,30,50],
230.        #     'n_estimators': [300, 400, 1000]
231.        # }
232.
233.        # # Creación del modelo
234.        # rf = RandomForestClassifier()
235.
236.        # # Instanciamos la búsqueda de rejilla:
237.        # grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
238.        #                            cv = 4, n_jobs = -
    1, verbose = 2, scoring='f1_macro')
239.
240.        # grid_search.fit(X_train, Y_train)
241.
242.        # print('La combinación de parámetros que mejor métrica objetivo arroja es: ')
243.        # print(grid_search.best_params_)
244.        # print('El valor del F1-
    Score con la mejor combinación de parámetros es: ', grid_search.best_score_)
```

```
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

preprocessing.py

Este código es una adaptación a la UNSW-NB15 del desarrollado en
https://github.com/imoken1122/Intrusion-Detection-CVAE por el usuario
imoken1122. El código original toma como referencia el artículo “Conditional
Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion
Detection in IoT”, de Manuel López-Martín, Belén Carro, Antonio Sánchez-Esguevillas
y Jaime Lloret (2017), que trabaja con la base de datos NSL-KDD.

En preprocessing.py se realiza el preprocesado y preparación de los conjuntos de
entrenamiento y test de la base de datos UNSW-NB15 para su posterior
clasificación binaria o multiclase por parte del CVAE.
"""

# ============================================================================
#                            IMPORTS DE LIBRERÍAS
# ============================================================================

import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler


# ============================================================================
#                                  INICIO
# ============================================================================

print("Para preparar la base de datos UNSW-
NB15 para una clasificación multiclase, inserte 0.\n")
print("Para preparar la base de datos UNSW-
NB15 para una clasificación binaria, inserte 1.\n")
#flag para etiquetar las clases de acuerdo a una clasificación binaria o multiclase.
BINARIO = input("Inserte un número: ")
print("\n")

while BINARIO != '0' and BINARIO != '1':
    print("VALOR NO RECONOCIDO.\n")
    print("Para preparar la base de datos UNSW-
NB15 para una clasificación multiclase, inserte 0.\n")
    print("Para preparar la base de datos UNSW-
NB15 para una clasificación binaria, inserte 1.\n")
    BINARIO = input("Inserte un número: ")
    print("\n")



# importamos los dos CSV para preprocesarlos.
train = pd.read_csv("dataset/UNSW_NB15_training-set.csv", header=0)
test = pd.read_csv("dataset/UNSW_NB15_testing-set.csv", header=0)


# Descartamos features que en su mayoría son 0 o no sirven:
discard = ["id","is_ftp_login","ct_ftp_cmd","trans_depth","is_sm_ips_ports","ct_flw_htt
p_mthd"]
for i in discard:
    del train[i]
    del test[i]

#dummy_c guarda las variables categóricas:
dummy_c = ["proto","service","state"]


# ============================================================================
```

```
63. #                           NORMALIZACIÓN MIN-MAX
64. # ============================================================================
65. 
66. # numeric_columns guarda las features numéricas del conjunto de datos:
67. numeric_columns = list(train.select_dtypes(include=['int64','float64']).columns)
68. minmax = MinMaxScaler()
69. 
70. # Aplicación de Min-Max:
71. for c in numeric_columns:
72.     train[c] = minmax.fit_transform(np.array(train[c]).reshape(-1,1))
73.     test[c] = minmax.fit_transform(np.array(test[c]).reshape(-1,1))
74. 
75. # ============================================================================
76. #              DIFERENCIA DE ETIQUETAS EN SETS DE TRAIN Y TEST
77. # ============================================================================
78. 
79. # Completamos con columnas vacías las categorías faltantes en ambos conjuntos.
80. 
81. def labels(tra,tes,feat):
82.     trainset=tra[feat].tolist()
83.     testset= tes[feat].tolist()
84.     diff1=list(set(trainset) - set(testset))
85.     diff2=list(set(testset) - set(trainset))
86.     return(diff1, diff2)
87. 
88. 
89. for feat in dummy_c:
90. 
91.     dif1,dif2 = labels(train, test, feat)
92.     print('FEATURE ' + feat + '. Faltan en el set de testing: ')
93.     print(dif1)
94.     if len(dif1) != 0:
95.         print('Completamos el set de test con las categorías que faltan...')
96.         for ind in range(len(dif1)):
97.             # cambiamos esta línea de como estaba originalmente para que el get_dummies
     del model.py
98.             # opere correctamente (get_dummies pone primero como nombre la categoría a
    la que se le aplica OH
99.             # y luego el nombre de la característica.)
100.                 test.insert(len(test.columns),feat + "_" + dif1[ind],0)
101. 
102. 
103.         print('FEATURE ' + feat + '. Faltan en el set de entreno: ')
104.         print(dif2)
105.         if len(dif2) != 0:
106.             print('Completamos el set de entreno con las categorías que faltan...')

107.             for ind in range(len(dif2)):
108.                 train.insert(len(train.columns),feat + "_" + dif2[ind],0)
109.             print("\n")
110. 
111.     print("\n")
112. 
113. 
114. 
115.     # ============================================================================

116.     #                           CODIFICACIÓN ONE-HOT
117.     # ============================================================================

118. 
119.     # Aplicación de codificación OH a las features categóricas
120.     train = pd.get_dummies(train,columns=dummy_c)
121.     test = pd.get_dummies(test,columns=dummy_c)
122. 
123. 
```

```
124.        # ================================================================================
125.        #                  GUARDADO DE CONJUNTOS DE DATOS EN .CSV
126.        # ================================================================================
127.
128.        # Si queremos preparar la bbdd para clasif. binaria, mantenemos la columna label
129.        # y borramos attack_cat.
130.
131.        if BINARIO == '1':
132.            test = test.drop(['attack_cat'],axis='columns')
133.            train = train.drop(['attack_cat'],axis='columns')
134.
135.            # Renombramos label como class para facilitar la ejecución
136.            # en el resto de códigos:
137.            train = train.rename(columns={'label':'class'})
138.            test = test.rename(columns={'label':'class'})
139.
140.            train.to_csv("dataset/VAE_Train+.csv", index=False)
141.            test.to_csv("dataset/VAE_Test+.csv", index=False)
142.
143.            # Si queremos preparar la bbdd para clasif. multiclase, mantenemos
144.            #la columna attack_cat y borramos label.
145.        else:
146.            test = test.drop(['label'],1)
147.            train = train.drop(['label'],1)
148.
149.            # #mapeo de las distintas clases a un número para facilitar la codif. OH:
150.            mapeo = ({"Normal":0, "Generic":1,"Exploits":2,"Fuzzers":3,"DoS":4,
151.                      "Reconnaissance":5,"Analysis":6, "Backdoor":7,"Shellcode":8,"Worms
    ":9})
152.            train["attack_cat"] = train["attack_cat"].map(lambda x : mapeo[x])
153.            test["attack_cat"] = test["attack_cat"].map(lambda x : mapeo[x])
154.
155.            # Renombramos attack_cat como class para facilitar la ejecución
156.            # en el resto de códigos:
157.            train = train.rename(columns={'attack_cat':'class'})
158.            test = test.rename(columns={'attack_cat':'class'})
159.
160.            train.to_csv("dataset/VAE_Train_M+.csv", index=False)
161.            test.to_csv("dataset/VAE_Test_M+.csv",index=False)
```

```
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

model.py

Este código es una adaptación a la UNSW-NB15 del desarrollado en
https://github.com/imoken1122/Intrusion-Detection-CVAE por el usuario
imoken1122. El código original toma como referencia el artículo “Conditional
Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion
Detection in IoT”, de Manuel López-Martín, Belén Carro, Antonio Sánchez-Esguevillas
y Jaime Lloret (2017), que trabaja con la base de datos NSL-KDD.

En model.py se define la estructura de capas neuronales del CVAE por las que
pasan las instancias. Este modelo es invocado en cvae.py y valuate.py, para
realizar el entrenamiento y clasificación de las muestras, respectivamente.
"""

# ============================================================================
#                            IMPORTS DE LIBRERÍAS
# ============================================================================
import torch as th
from torch import nn
from torch.autograd import Variable as V


class CVAE(nn.Module):
    def __init__(self):
        super().__init__()

# ============================================================================
#                       DEFINICIÓN DE CAPAS DEL MODELO
# ============================================================================
        # 191: nº de features de nuestra bbdd (sin la feature 'class')
        self.fc1=nn.Linear(191,500)
        self.hidden = nn.Linear(500,500)
        self.mu = nn.Linear(500,25)
        self.sigma = nn.Linear(500,25)


        #la salida de fc2 será 498 (si clasificación binaria, para concatenar
        # las 2 etiquetas de clase (Ataque-No Ataque) y que concuerde con la
        # entrada de la siguiente capa; o 490 (si clasif. multiclase), para
        # concatenar las 10 etiquetas de clase en el decoder.

        # self.fc2 = nn.Linear(25,498) #SI SE QUIERE CLASIF. BINARIA
        self.fc2 = nn.Linear(25,490) #SI SE QUIERE CLASIF. MULTICLASE
        self.fc3 = nn.Linear(500,191)
        self.sigmoid = nn.Sigmoid()
        self.relu = nn.ReLU()



# ============================================================================
# Función encoder: detalla la estructura de capas del codificador del CVAE
#                  Q(z|X) por las que pasarán las instancias de la UNSW-
NB15.
#
#       Entradas: x: conjunto de instancias pertenecientes a un batch de la
#                    UNSW-NB15.
#
#       Salidas:  mu: media de la dist. Gaussiana Q(z|X)
#                 sigma: desv. típica de la dist. Gaussiana Q(z|X)
#
# ============================================================================

    def encoder(self,x):
```

```
67.         h = self.relu(self.fc1(x))
68.         h = self.relu(self.hidden(h))
69.         h = self.hidden(h)
70.         return self.mu(h),self.sigma(h)
71.
72. # ================================================================================
73. # Función reparametrization: Se aplica para poder aplicar backpropagation en el
74. #                            modelo y actualizar los pesos de éste.
75. #                            Se desvía el comportamiento no diferenciable fuera de
76. #                            la red, ya que para un para un proceso de muestreo
77. #                            aleatorio no se puede aplicar
78. #                            la propagación hacia atrás.
79. #
80. #
81. #       Entradas: mu: media de la dist. Gaussiana Q(z|X)
82. #                 logsigma: desv. típica de la dist. Gaussiana Q(z|X)
83. #
84. #       Salidas:  sigma.mul(eps) + mu (ya que la distrib. del decoder es una normal)
85. #
86. #
87. # ================================================================================
88.     def reparametrization(self,mu,logsigma):
89.         sigma = th.exp(0.5*logsigma)
90.         eps = V(th.randn(sigma.size()))
91.         return sigma.mul(eps) + mu
92.
93.
94. # ================================================================================
95. # Función decoder: detalla la estructura de capas del decodificador del CVAE
96. #                  P(X|z,L) por las que pasarán las instancias de la UNSW-
    NB15.
97. #
98. #       Entradas: z: distribución normal del espacio latente.
99. #                 oh_label: etiqueta de clase (en formato One-Hot)
100.       #
101.       #       Salidas:  output: instancias reconstruidas por el modelo.
102.       #
103.       #
104.       # ================================================================================

105.           def decoder(self,z,oh_label):
106.               h = self.relu(self.fc2(z))
107.               #concatenación de la etiqueta de clase (en formato One-
    Hot) a las features.
108.               h = th.cat((h,oh_label),dim = 1)
109.               h = self.fc3(self.relu(self.hidden(h)))
110.               return self.sigmoid(h)
111.
112.
113.       # ================================================================================

114.       # Función forward: llama a todas las funciones anteriores, devolviendo la salida

115.       #                  arrojada por el modelo una vez ha procesado todos los datos.

116.       #
117.       #
118.       #       Entradas: x: conjunto de instancias pertenecientes a un batch de la
119.       #                    UNSW-NB15.
120.       #                 label: etiquetas de clase (en formato One-Hot)
121.       #
122.       #       Salidas:  output: instancias reconstruidas por el modelo.
123.       #                 mu: media de la dist. Gaussiana Q(z|X)
124.       #                 sigma: desv. típica de la dist. Gaussiana Q(z|X)
125.       # ================================================================================

126.           def forward(self,x,label):
```

```
127.                mu,sigma = self.encoder(x)
128.                z = self.reparametrization(mu,sigma)
129.                output = self.decoder(z,label)
130.                return output,mu,sigma
```

```python
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

cvae.py

Este código es una adaptación a la UNSW-NB15 del desarrollado en
https://github.com/imoken1122/Intrusion-Detection-CVAE por el usuario
imoken1122. El código original toma como referencia el artículo “Conditional
Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion
Detection in IoT”, de Manuel López-Martín, Belén Carro, Antonio Sánchez-Esguevillas
y Jaime Lloret (2017), que trabaja con la base de datos NSL-KDD.

En cvae.py se realiza el entrenamiento (configuración de pesos y bias del modelo)
y cálculo de la función de pérdida para los conjuntos de entrenamiento y test
para un número de iteraciones determinado. Los conjuntos de datos utilizados son
los generados en el fichero proprocessing.py, pertenecientes a la base de datos
UNSW-NB15.
Una vez entrenado el modelo, se guarda para poder ser utilizado en el fichero
valuate.py, el encargado de realizar la clasificación binaria o multiclase con el
conjunto de test.
"""

# ============================================================================
#                            IMPORTS DE LIBRERÍAS
# ============================================================================

import pandas as pd
import numpy as np
import torch as th
from torch import optim
from torch.autograd import Variable as V
from torch.nn import functional as F
import matplotlib.pyplot as plt
from model import CVAE


# ============================================================================
#          DEFINICIÓN DE TIPO DE ENTRENAMIENTO Y VARIABLES E
#                   IMPORTACIÓN DE CONJUNTOS DE DATOS
# ============================================================================


print("Para realizar un entrenamiento multiclase, inserte 0.\n")
print("Para realizar un entrenamiento binario, inserte 1.\n")

#flag para cargar el conjunto de datos adecuado, dependiendo de si el entrenamiento
#es binario (1) o multiclase (0).
BINARIO = input("Inserte un número: ")
print("\n")

while BINARIO != '0' and BINARIO != '1':
    print("VALOR NO RECONOCIDO.\n")
    print("Para realizar un entrenamiento multiclase, inserte 0.\n")
    print("Para realizar un entrenamiento binario, inserte 1.\n")
    BINARIO = input("Inserte un número: ")
    print("\n")


# Si clasificacion binaria:
if BINARIO == '1':
    train = pd.read_csv("dataset/VAE_Train+.csv")
    test = pd.read_csv("dataset/VAE_Test+.csv")
    nombre="binario" #variable para ponerle nombre al modelo una vez se guarde.
# Si clasificacion multiclase:
else:
    train = pd.read_csv("dataset/VAE_Train_M+.csv")
```

```
    test = pd.read_csv("dataset/VAE_Test_M+.csv")
    nombre="multiclase"

# Separación de los conjuntos de datos en instancias y etiquetas de clase.
trainx, trainy = (np.array(train[train.columns[train.columns != "class"]]),
                  np.array(pd.get_dummies(train["class"])))
testx, testy= (np.array(test[train.columns[train.columns != "class"]]),
               np.array(pd.get_dummies(test["class"])))

batch_size = 512 #tamaño de los batches en los que se dividirán los conjuntos de datos
max_epoch = 100 #epochs del entrenamiento
train_N = len(train) #instancias del conjunto de entrenamiento
test_N = len(test)  # instancias del conjunto de test


# creamos una instancia del modelo CVAE:
model = CVAE()

# Como optimizador del algoritmo elegimos Adadelta.
opt = optim.Adadelta(model.parameters(),lr = 1e-3)
# ================================================================================
#                          DEFINICIÓN DE FUNCIONES
# ================================================================================

# ================================================================================
#  Función Loss_function: Calcula la función de pérdida (ELBO) del autocodificador
#                        variacional condicional. Consta de dos términos:
#
#           KL_div: término de regularización. se mide a través de la divergencia KL
#           y trata de asegurar que las variables contenidas en el espacio latente
#           describen de forma eficiente los datos de entrada.
#
      #           reconstruction_loss: pérdida generativa o error de reconstrucción.
      #         Intenta medir la eficacia conjunta del codificador y el decodificador
      #          con respecto a los datos iniciales, esto es, cuantificar cómo de bien
      #          reconstruye el modelo las entradas a partir de las salidas obtenidas.
      #
      #
      #       Entradas: x_hat: conjunto de datos reconstruido por el CVAE.
      #                 x: conjunto de datos original
      #                 mu: media de la dist. Gaussiana Q(z|X)
      #                 logsigma: desv. típica de la dist. Gaussiana Q(z|X)
      #
      #       Salidas: reconstruction_loss: E[log P(X|z)]
      #                 KL_div: D_KL(Q(z|X) || P(z|X)); calculado de forma cerrada ya
      #                  que ambas distancias son distribuciones Gaussianas.
      # ================================================================================

      def Loss_function(x_hat,x, mu,logsigma):
        #Se utiliza la entropía cruzada binaria en la función de pérdida porque
        #todos los valores están escalados entre [0-1].
        reconstruction_loss = F.binary_cross_entropy(x_hat,x,size_average = False)
        # Al ser Q(z|X) y P(z) Gaussianas, se puede computar la div. KL así:
        KL_div = -0.5 * th.sum(1+logsigma-mu.pow(2) - logsigma.exp())
        return reconstruction_loss+KL_div


      # ================================================================================

      #  Función create_batch: Divide el conjunto de entrenamiento (o de test) y
```

```
127.      #                     las etiquetas de las clases en batches para ir entrenando
128.      #                        el modelo con batches.
129.      #
130.      #
131.      #       Entradas: x: conjunto de datos de entrenamiento (o test)
132.      #                 y: etiquetas de clase del conj. de datos de entrenamiento (o t
          est)
133.      #
134.      #       Salidas: batch_x: instancias del conj. de entrenamiento (o test)
135.      #                         agrupadas en batches
136.      #                batch_y: etiquetas de clase del conj. de entrenamiento (o test)
137.      #                          agrupadas en batches
138.      #
139.      # ================================================================================

140.
141.      def create_batch(x,y):
142.          # aleatorizamos el conjunto de datos
143.          a = list(range(len(x)))
144.          np.random.shuffle(a)
145.          x = x[a]
146.          y = y[a]
147.          #dividimos las instancias y sus correspondientes etiquetas de clase en
148.          # batches de tamaño batch_size
149.          batch_x = ([x[batch_size * i : (i+1)*batch_size,:].tolist()
150.                      for i in range(len(x)//batch_size)])
151.
152.          batch_y = ([y[batch_size * i : (i+1)*batch_size].tolist()
153.                      for i in range(len(x)//batch_size)])
154.          return batch_x, batch_y
155.
156.
157.      # ================================================================================

158.      #  Función train: Aplicación del modelo CVAE al conj. de entrenamiento. Se llama
159.      #             a la función create_batch para crear los batches que se le pasarán
160.      #                al modelo. Después se calcula la función de pérdida y se aplica
161.      #                 backpropagation para calcular los gradientes del modelo
162.      #                 y posteriormente actualizar sus pesos.
163.      #
164.      #
165.      #       Entradas: --
166.      #
167.      #       Salidas: tr_loss/train_N: cociente entre el total acumulado de pérdidas
168.      #                del epoch y el número de instancias del conj. de entrenamiento.
169.      #
170.      #
171.      # ================================================================================

172.
173.      def train():
174.          model.train() #se activa el entrenamiento del modelo.
175.          tr_loss = 0 #variable que guarda el acumulado de las pérdidas para todo el
176.                      #conjunto de datos.
177.
178.          #dividimos el conj. de entrenamiento en batches (instancias y etiquetas)
179.          batch_x,batch_y = create_batch(trainx,trainy)
180.          for x,y in zip(batch_x,batch_y):
181.              #borramos la información de los gradientes del batch previo
182.              opt.zero_grad()
```

```
183.
184.              #Creación de tensores para las instancias y las etiquetas. Las
185.              #metemos en Variables para poder operar con ellas (aplicar gradientes)
186.              #en el modelo del CVAE.
187.              x,y = V(th.Tensor(x)),V(th.Tensor(y))
188.
189.              #se alimenta el modelo con el conj. de datos de entrenamiento.
190.              x_hat,mu,logsigma = model(x,y)
191.
192.              #cálculo del valor de pérdida para el batch
193.              loss = Loss_function(x_hat,x,mu,logsigma)
194.
195.              # backward pass
196.              loss.backward() #cálculo de gradiente del tensor de pérdida
197.              tr_loss += loss.item() #sumamos la pérdida del batch al total
198.              opt.step() #actualización de pesos del modelo
199.          return tr_loss/train_N
200.       # ========================================================================
201.      #  Función test: Aplicación del modelo CVAE al conjunto de test. Se llama
202.      #                 a la función create_batch para crear los batches que se le
    pasarán
203.      #                 al modelo. Después se calcula la función de pérdida
204.      #
205.      #
206.      #       Entradas: --
207.      #
208.      #       Salidas: te_loss/test_N: cociente entre el total acumulado de pérdidas
209.      #                del epoch y el número de instancias del conj. de test.
210.      #
211.      #
212.      # ========================================================================
213.
214.      def test():
215.          model.eval() #se activa la evaluación del modelo.
216.          te_loss = 0 #variable que guarda el acumulado de las pérdidas para todo el
217.                      #conjunto de datos.
218.
219.          #dividimos el conj. de test en batches (instancias y etiquetas)
220.          batch_x,batch_y = create_batch(testx,testy)
221.
222.          with th.no_grad(): #se desactiva el cálculo de gradientes
223.              for x,y in zip(batch_x,batch_y):
224.
225.                  #Creación de tensores para las instancias y las etiquetas. Las
226.                  #metemos en Variables para poder operar con ellas (aplicar gradientes)
227.                  #en el modelo del CVAE.
228.                  x,y = V(th.Tensor(x)),V(th.Tensor(y))
229.
230.                  #se alimenta el modelo con el conj. de datos de test.
231.                  x_hat,mu,sigma = model(x,y)
232.
233.                  #cálculo del valor de pérdida para el batch
234.                  loss = Loss_function(x_hat,x,mu,sigma)
235.                  te_loss += loss.item() #sumamos la pérdida del batch al total
236.
237.          return te_loss/test_N
238.
239.      # ========================================================================
240.      #                                 INICIO
241.      # ========================================================================

242.
243.      tr_loss ,te_loss = [],[] #vectores donde se guardan los valores de pérdida
244.                               #de cada epoch para entrenamiento y test.
245.
246.      for epoch in range(max_epoch):
247.          trl = train()
```

```
248.            tel = test()
249.            #se guardan en los vectores los valores de pérdida para el epoch actual
250.            tr_loss.append(trl)
251.            te_loss.append(tel)
252.            if epoch % 2 == 0:
253.                print(epoch,trl,tel)
254.
255.        # Guardado del modelo
256.        th.save(model.state_dict(),f"save_model/cvae_adadelta_{max_epoch}_{nombre}.pth")
257.
258.        #Representación de los valores de pérdida frente a los epochs para
259.        #entrenamiento y test
260.        plt.subplots(figsize = (8,6))
261.        plt.title("Función de pérdida vs. epochs (binario)")
262.        plt.xlabel('Epoch')
263.        plt.ylabel('F.pérdida')
264.        trainloss = plt.plot(tr_loss, label='Entrenamiento')
265.        testloss = plt.plot(te_loss, label='Test')
266.        plt.legend()
267.        plt.show()
```

```
# -*- coding: utf-8 -*-
"""
@author: María Dolores Trapero Estepa

valuate.py

Este código es una adaptación a la UNSW-NB15 del desarrollado en
https://github.com/imoken1122/Intrusion-Detection-CVAE por el usuario
imoken1122. El código original toma como referencia el artículo “Conditional
Variational Autoencoder for Prediction and Feature Recovery Applied to Intrusion
Detection in IoT”, de Manuel López-Martín, Belén Carro, Antonio Sánchez-Esguevillas
y Jaime Lloret (2017), que trabaja con la base de datos NSL-KDD.

En valuate.py se realiza la clasificación (binaria o multiclase) del conjunto
de test de la base de datos UNSW-NB15. Para cada instancia, se computa la distancia
entre la muestra original(x) y la reconstruida por el modelo(x_rec),
y se repite esto para todas las etiquetas de clase. La clase que arroje menor
distancia entre muestra original y reconstruida por el CVAE se escoge
como clase predecida.
"""

# ============================================================================
#                             IMPORTS DE LIBRERÍAS
# ============================================================================

import torch as th
import pandas as pd
from model import CVAE
import numpy as np
from metricas import *
from time import time

# ============================================================================
#                          DEFINICIÓN DE FUNCIONES
# ============================================================================

# ============================================================================
# Función reverse_OH: Deshace la codificación one-hot cambiando los vectores
#                     de 0's y 1's por las etiquetas categóricas adecuadas.

#
#       Entradas: vector: matriz de 0's y 1's a la que se le quiere deshacer
#                 la codificación OH.
#                 etiquetas: etiquetas de clase (binario o multiclase)
#
#       Salidas:  vector_cat: vector con etiquetas categóricas en lugar de OH.
#
# ============================================================================

def reverse_OH(vector, etiquetas):
    vector_cat=[] #guarda la etiqueta de clase a la que pertenece la predicción
    #el bucle recorre todas las filas de las predicciones
    for c in range(vector.shape[0]):
        #buscamos el índice del elemento distinto de cero de la fila c de
        #vector:
        arr=np.nonzero(vector[c])

        #se busca en labels la etiqueta que corresponde al índice != 0
        vector_cat.append(labels[int(arr[0])])
    return vector_cat

# ============================================================================
#                                 INICIO
# ============================================================================

# creamos una instancia del modelo CVAE:
model = CVAE()
```

```
67.
68. print("Para realizar una clasificación multiclase, inserte 0.\n")
69. print("Para realizar una clasificación binaria, inserte 1.\n")
70.
71. #flag para cargar el conjunto de datos adecuado, dependiendo de si la clasif.
72. #es binaria (1) o multiclase (0).
73. BINARIO = input("Inserte un número: ")
74. print("\n")
75.
76. while BINARIO != '0' and BINARIO != '1':
77.     print("VALOR NO RECONOCIDO.\n")
78.     print("Para realizar un entrenamiento multiclase, inserte 0.\n")
79.     print("Para realizar un entrenamiento binario, inserte 1.\n")
80.     BINARIO = input("Inserte un número: ")
81.     print("\n")
82.
83. if BINARIO == '1':
84.     test = pd.read_csv("dataset/VAE_Test+.csv")
85.     param = th.load('save_model/cvae_adadelta_100_binario.pth',map_location=lambda x,y:
    x)
86.     labels = np.array(test['class'].unique(), dtype=np.uint8)
87. else:
88.     test=pd.read_csv("dataset/VAE_Test_M+.csv")
89.     param = th.load('save_model/cvae_adadelta_100_multiclase.pth',map_location=lambda x
    ,y:x)
90.     labels=(["Normal", "Generic","Exploits","Fuzzers","DoS","Reconnaissance",
91.                     "Analysis", "Backdoor","Shellcode","Worms"])
92.
93. # Separación de los conjuntos de datos en datos y etiquetas de clase.
94. testx, testy= (np.array(test[test.columns[test.columns != "class"]]),
95.                np.array(pd.get_dummies(test["class"])))
96.
97. #cargamos los parámetros (pesos y sesgos) del modelo entrenado en cvae.py
98. #a una nueva instancia del modelo.
99. model.load_state_dict(param)
100.
101.
102.       n,m = testx.shape[1],testy.shape[1]
103.       test_label = th.eye(m) #matriz identidad de mxm
104.       pred = [] #guarda la estimación del CVAE clasificando las muestras del conj. tes
    t.
105.
106.       tiempo_inicial = time()
107.       for x in testx:
108.           #each_dist guarda la distancia entre x y x_rec, calculadas para todas las cl
    ases
109.           each_dist = []
110.
111.           x = th.Tensor(x.reshape(1,n))
112.
113.           # Para cada instancia, se computa la distancia entre la muestra original y
114.           # la muestra reconstruida por el CVAE para todas las etiquetas de clase.
115.           # P.ej: si la clasificación es binaria, para una misma instancia se computan
     las
116.           # distancias para la clase "Attack" y para la clase "Normal". De ambas, la m
    enor
117.           # distancia se escoge como clase predecida por el modelo.
118.
119.           for label in test_label:
120.               label = th.Tensor(label.reshape(1,m))
121.
122.               #pasamos al modelo una instancia y una etiqueta de clase.
123.               x_hat,mu,sigma = model(x,label)
124.
125.               #cálculo de la distancia Euclídea entre la instancia original
126.               # y la reconstruida para una clase en concreto.
127.               # cdist hace el sumatorio de la dist. Euclídea entre pares:
```

```
128.                # Para P1=(x1,y1) y P2=(x2,y2), dist=sqrt((x2-x1)^2+(y2-y1)^2)
129.                dist = th.cdist(x_hat,x,p=2)
130.                each_dist.append(dist) #se guarda la distancia para esa clase en particu
    lar
131.
132.            #se guarda en pred la clase predecida para esa muestra, que se corresponde
133.            #con la que tiene menor distancia
134.            pred.append(np.identity(m)[np.argmin(each_dist)])
135.
136.        tiempo_final = time()
137.        # ================================================================================

138.        #                      RESULTADOS DE CLASIFICACIÓN
139.        # ================================================================================

140.
141.        #convertimos a uint8 y array numpy las clases predecidas por el CVAE
142.        pred=np.array(pred, dtype=np.uint8)
143.
144.        #Deshacemos la codificación one-hot de las etiquetas de clase de las muestras
145.        #para poder graficar la matriz de confusión con la función reverse_OH.
146.        y_test=reverse_OH(testy,labels)
147.        preds=reverse_OH(pred,labels)
148.
149.        tiempo_ejecucion = tiempo_final - tiempo_inicial
150.        print('El tiempo de ejecución fue: %0.3f segundos.' % tiempo_ejecucion)
151.
152.        performance(y_test,preds,labels)
```

```python
# -*- coding: utf-8 -*-

"""
@author: María Dolores Trapero Estepa

mlp.py

Realiza el preprocesado, entrenamiento y posterior clasificación binaria y
multiclase de la base de datos UNSW-NB15 con un perceptrón multicapa, siguiendo
la idea del artículo "Evaluation of Machine Learning Algorithms for
Anomaly Detection", de Nebrase Elmbrabit, Feixiang Zhou, Fengyin Li y Huiyu Zhou.
"""


# Imports de librerías:
import pandas as pd
from metricas import *
from sklearn.preprocessing import MinMaxScaler
from time import time
from sklearn.compose import ColumnTransformer
from sklearn.neural_network import MLPClassifier


# ============================================================================
#           DIVISIÓN DE DATOS EN SETS DE TRAIN Y TEST
# ============================================================================

# importamos los dos CSV para preprocesarlos.
train = pd.read_csv("UNSW_NB15_training-set.csv", header=0)
test = pd.read_csv("UNSW_NB15_testing-set.csv", header=0)


# Se verifica que no hay valores nulos en el dataset.
print(train.isnull().values.any()) # Si False, no hay valores nulos.
print(test.isnull().values.any()) # Si False, no hay valores nulos.

# Elegimos qué tipo de clasificación queremos realizar:
print("Para realizar clasificación multiclase, inserte 0.\n")
print("Para realizar clasificación binaria, inserte 1.\n")
datos = input("Inserte un número: ")
print("\n")

while datos != '0' and datos != '1':
    print("VALOR NO RECONOCIDO.\n")
    print("Para realizar clasificación multiclase, inserte 0.\n")
    print("Para realizar clasificación binaria, inserte 1.\n")
    datos = input("Inserte un número: ")
    print("\n")

if datos == '1':
    print("CLASIFICACIÓN BINARIA.\n")
    # Separamos la característica label del resto de características.
    # Lo hacemos para el conjunto de test y para el de training.
    Y_train = train['label']
    Y_test =  test['label']
    X_train = train.drop(['attack_cat', 'label'],axis='columns')
    X_test = test.drop(['attack_cat','label'],axis='columns')
    act='logistic'

else:
    print("CLASIFICACIÓN MULTICLASE.\n")
    # Separamos la característica attack_cat del resto de características.
    # Lo hacemos para el conjunto de test y para el de training.
    Y_train = train['attack_cat']
    Y_test =  test['attack_cat']
    X_train = train.drop(['label','attack_cat'],axis='columns')
    X_test = test.drop(['label','attack_cat'],axis='columns')
```

```
    act='relu'


# Distribución de variables categóricas en train y test.
features=['proto','service','state']

#Aplicamos feature selection escogiendo las features más importantes
#destacadas en el artículo:

X_train=(X_train[['sttl','ct_srv_dst','sbytes','smean','proto','ct_state_ttl',
                  'sloss','synack','ct_dst_src_ltm','dmean','ct_srv_src',
                  'service','ct_dst_sport_ltm','dbytes','dloss','state',
                  'tcprtt','ct_src_dport_ltm','rate']])

X_test=(X_test[['sttl','ct_srv_dst','sbytes','smean','proto','ct_state_ttl',
                  'sloss','synack','ct_dst_src_ltm','dmean','ct_srv_src',
                  'service','ct_dst_sport_ltm','dbytes','dloss','state',
                  'tcprtt','ct_src_dport_ltm','rate']])

# ================================================================================
#                            NORMALIZACIÓN
# ================================================================================

# Eliminamos las vbles categóricas de las columnas para poder quedarnos con las
# vbles numéricas (a las que aplicaremos la normalización)
numcol=X_train.columns
numcol=numcol.drop(features)
numcol=pd.Index(numcol)
numcol=numcol.to_list()

# guardamos el nombre de las columnas y el tipo de datos para despues convertir a DataF
rame
# y seguir procesando los datos de esta forma.
      listacols=numcol + features
      types=X_train.dtypes


      # reordenamos los tipos de datos para asociarlos a las features adecuadas
      types= types.reindex(listacols)

      # normalizamos solo las categorías numéricas:
      sca=MinMaxScaler()

      # con passthrough hacemos que el resto de columnas las deje como está
      column_trans = ColumnTransformer([('scaler', sca, numcol)],remainder='passthroug
h')

      X_test= column_trans.fit_transform(X_test)

      X_train = column_trans.fit_transform(X_train)

      # reconvertimos a DataFrame
      X_train=pd.DataFrame(X_train,columns=listacols)
      X_test=pd.DataFrame(X_test,columns=listacols)


      # ================================================================================

      #             DIFERENCIA DE ETIQUETAS EN SETS DE TRAIN Y TEST
      # ================================================================================


      # Completamos con columnas vacías las categorías faltantes en ambos conjuntos.

      def labels(tra,tes,feat):
          trainset=tra[feat].tolist()
          testset= tes[feat].tolist()
```

```
131.            diff1=list(set(trainset) - set(testset))
132.            diff2=list(set(testset) - set(trainset))
133.            return(diff1, diff2)
134.
135.
136.        for feat in features:
137.
138.            dif1,dif2 = labels(X_train, X_test, feat)
139.            print('FEATURE ' + feat + '. Faltan en el set de testing: ')
140.            print(dif1)
141.            if len(dif1) != 0:
142.                print('Completamos el set de test con las categorías que faltan...')
143.                for ind in range(len(dif1)):
144.                    X_test.insert(len(X_test.columns),dif1[ind],0)
145.
146.
147.            print('FEATURE ' + feat + '. Faltan en el set de entreno: ')
148.            print(dif2)
149.            if len(dif2) != 0:
150.                print('Completamos el set de entreno con las categorías que faltan...')
151.                for ind in range(len(dif2)):
152.                    X_train.insert(len(X_train.columns),dif2[ind],0)
153.                print("\n")
154.
155.        print("\n")
156.
157.
158.
159.        # ================================================================================
160.        #                              ONE-HOT ENCODING
161.        # ================================================================================
162.
163.        # Creamos una función para aplicar a todas las variables categóricas:
164.        def onehot(dataframe_orig, feat_a_codif):
165.            dummies = pd.get_dummies(dataframe_orig[feat_a_codif])
166.            #añadimos al array original la codificación
167.            res = pd.concat([dataframe_orig, dummies], axis=1)
168.            #tiramos la columna de la feature convertida
169.            res = res.drop([feat_a_codif], axis=1)
170.            return(res)
171.
172.
173.        # Identificamos las features categóricas para poder convertirlas a enteros:
174.        print('Train set:')
175.        for col_name in X_train.columns:
176.            if col_name == 'proto' or col_name == 'service' or col_name == 'state' :
177.                unique_cat = len(X_train[col_name].unique())
178.                (print("El feature '{col_name}' tiene {unique_cat} categorías"
179.                       .format(col_name=col_name, unique_cat=unique_cat)))
180.                X_train=onehot(X_train, col_name)
181.
182.        print("\n")
183.
184.        print('Test set')
185.        for col_name in X_test.columns:
186.            if col_name == 'proto' or col_name == 'service' or col_name == 'state' :
187.                unique_cat = len(X_test[col_name].unique())
188.                (print("El feature '{col_name}' tiene {unique_cat} categorías"
189.                       .format(col_name=col_name, unique_cat=unique_cat)))
190.                X_test=onehot(X_test, col_name)
191.
192.        print("\n")
193.
194.
```

```
195.      # ================================================================================
196.      #                        CLASIFICACIÓN CON ALGORITMOS
197.      # ================================================================================
198.
199.      # Etiquetas para matriz de confusión
200.      labels = Y_test.unique()
201.
202.      # Entrenamos el MLP
203.      # Función de activación por: sigmoide (logistic) (clasif. binaria)
204.      #                            relu (clasif. multiclase)
205.      # Iteraciones por defecto: 200
206.      mlp_clf = (MLPClassifier(hidden_layer_sizes=(600,400,200,100,50),
207.                               learning_rate_init=0.002,activation=act,
208.                               learning_rate="adaptive",solver='adam',verbose=1))
209.
210.      tiempo_inicial = time()
211.      mlp_clf.fit(X_train, Y_train)
212.
213.      # Realizamos la clasificación con el set de test
214.      Y_pred = mlp_clf.predict(X_test)
215.      tiempo_final = time()
216.      tiempo_ejecucion = tiempo_final - tiempo_inicial
217.
218.      print('El tiempo de ejecución fue: %0.3f segundos.' % tiempo_ejecucion)
219.
220.      # Sacamos las métricas del clasificador:
221.      performance(Y_test,Y_pred,labels)
```