Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Detección de anomalías en la red empleando técnicas de machine learning

Estudiante: Julio Jairo Estévez Pereira
Dirección: Francisco Javier Nóvoa Manuel
Dirección: Diego Fernández Iglesias

A Coruña, septiembre de 2020.

A mi madre. Te dedico todo mi trabajo a ti, que fuiste mi talismán.

Agradecimientos

En primer lugar, quiero agradecer a mis tutores, Diego Fernández Iglesias y Francisco J. Nóvoa, que no sólo me hayan permitido trabajar con ellos, sino también guiado, y ayudado durante todo el proceso. También quiero dar las gracias a mis tíos, por todo el apoyo que me han brindado durante estos últimos años, sin el que no podría haber llegado hasta aquí. Por último, gracias Beth, por estar siempre ahí y no dejar que me salga del camino.

Resumen

En el ámbito de las redes de comunicación, una anomalía puede ser definida como una variación repentina del comportamiento habitual. Esto incluye tanto eventos fortuitos y bien intencionados, como ataques deliberados pensados para comprometer la disponibilidad de la red. En ambos casos, es esencial detectar rápido estas anomalías para poder reaccionar a tiempo.

En los últimos años, las técnicas de *machine learning* se han ido haciendo un lugar como alternativa a los sistemas tradicionales de detección de intrusiones basados en políticas, puesto que acarrean una serie de ventajas. Entre ellas, las técnicas de *machine learning* posibilitan el desarrollo de algoritmos no paramétricos, adaptativos a nuestra red y a sus modificaciones, y portables entre aplicaciones.

En este trabajo se revisan diferentes alternativas de *machine learning*, a medida que modelamos un *dataset* etiquetado con el tráfico de una red corporativa agregado en flujos de red. El proceso supone examinar minuciosamente el conjunto de datos, y seleccionar los más relevantes según nuestro dominio y las técnicas que empleamos. Después, buscamos la combinación idónea de parámetros para optimizar el rendimiento de los algoritmos y compararlos entre sí, de manera que podamos averiguar cuál es el mejor de forma objetiva.

También se barajan diferentes opciones en materia de computación distribuida para el despliegue, teniendo como objetivo la predicción en tiempo real de amenazas, y como requisito que sea posible el procesamiento de datos con características de *Big Data*. A fin de conseguir esto, se definen varias configuraciones de despliegue, y se emplean diferentes tecnologías que permiten emular el escenario descrito de predicción en tiempo real. En esta parte empleamos como infraestructura un *cluster* de diez máquinas. Por último, la solución que se despliega está basada en el mejor modelo obtenido.

Abstract

A network anomaly can be defined as a variation of the regular behaviour of the network. That includes both unfortunate unintended events, and deliberate attacks planned to compromise the network's availability. In both cases, it is essential to be able to detect those quickly so we can react in time.

In the past few years, machine learning has been slowly taking its place as an alternative to policies-based traditional intrusion detection systems, as it presents quite a few advantages. Amongst them, machine learning techniques allow the development of non-parametric algorithms, adaptative to our network and its modifications, and portable across applications.

In this paper we go through different machine learning and Big Data alternatives, as we model a labelled dataset containing the traffic of a corporative network aggregated in network flows. The process entails careful examination of the dataset, and the selection of the most relevant subset of data according to our domain and the employed techniques. Then we look for the best parameters combination to optimize the algorithms' performance, and we compare them to find out which one is the best in an objective way.

We also consider different options for deployment involving distributed computing, having as goal the prediction of network threats in real-time, and as requisite being able to process Big Data. For achieving this, we define several deployment configurations, and use different technologies so we can emulate the real-time prediction scenario that we just described. In this phase we rely on a cluster with ten machines as our infrastructure. Lastly, the solution deployed is based on the best model obtained.

Palabras clave:

- Machine learning
- IDS
- Seguridad de red
- Computación distribuida
- Big Data
- Flujo de red
- Anomalía
- Clasificación
- Random forest
- Naïve Bayes
- Deep Neural Network

Keywords:

- Machine learning
- IDS
- Network security
- Distributed computing
- Big Data
- Network flow
- Anomaly
- Classification
- Random forest
- Naïve Bayes
- Deep Neural Network

Índice general

1 Introducción **1**

1.1 Contextualización . . . 2

1.1.1 Anomalías, amenazas, y cómo detectarlas . . . 2

1.1.2 *Machine learning* en la detección de intrusiones . . . 4

1.1.3 *Big Data* y *machine learning* . . . 5

1.2 Objetivos . . . 8

1.3 Estructura del proyecto . . . 9

2 Metodología **11**

2.1 CRISP-DM . . . 11

3 Comprensión del negocio **15**

3.1 *Machine learning* . . . 16

3.1.1 Clasificación de técnicas . . . 17

3.1.2 *Deep Learning* . . . 18

3.2 Algoritmos . . . 20

3.2.1 *Naïve Bayes* . . . 20

3.2.2 *Random Forest* . . . 22

3.2.3 *Deep Neural Networks* . . . 23

3.3 *Dataset* . . . 25

3.3.1 UNB ISCXIDS2012 . . . 25

3.4 IDS y NTA . . . 26

3.5 Python/PySpark . . . 28

3.6 Scikit-learn . . . 28

3.7 Tensorflow y Keras . . . 29

3.8 Planificación del proyecto . . . 29

3.8.1 Metodología . . . 29

3.8.2 Planificación . . . 30

3.8.3 Seguimiento . . . 30
3.8.4 Estimación de costes . . . 31

4 Comprensión de los datos **33**
4.1 Recolección . . . 34
4.2 Exploración . . . 34
4.2.1 Grupo 1 . . . 35
4.2.2 Grupo 2 . . . 37
4.2.3 Grupo 3 . . . 38
4.2.4 Correlaciones . . . 39
4.3 Verificación de la calidad . . . 41
4.3.1 Datos erróneos . . . 41
4.3.2 Datos ausentes . . . 43

5 Preparación de los datos **45**
5.1 Limpieza . . . 46
5.1.1 *Isolation Forest* . . . 47
5.2 Transformación . . . 51
5.2.1 Cambio de formato . . . 51
5.2.2 Modificación de atributos . . . 52
5.2.3 Creación de atributos . . . 54
5.2.4 Escalado . . . 54
5.3 Selección . . . 55
5.3.1 Selección de muestras . . . 56
5.3.2 Selección de atributos . . . 57

6 Modelado **59**
6.1 Evaluación de modelos . . . 60
6.1.1 *Cross-Validation* . . . 60
6.1.2 Métricas de evaluación . . . 61
6.2 Construcción del modelo e hiperparametrización . . . 63
6.2.1 *Random Forest* . . . 63
6.2.2 *Naïve Bayes* . . . 65
6.2.3 *Deep Neural Network* . . . 67

7 Evaluación **69**
7.1 Evaluación de resultados . . . 70

8 Despliegue **71**
8.1 Plan de despliegue . . . 72
8.1.1 Creación del sistema . . . 72
8.1.2 Infraestructura . . . 73
8.2 Monitorización del sistema desplegado . . . 74
8.2.1 Mecanismos de monitorización . . . 74
8.2.2 Resultados del sistema . . . 75

9 Conclusiones y líneas futuras **77**

A Material adicional **81**
A.1 Repositorio . . . 81
A.2 Atributos del *dataset* UNB ISCX 2012 en detalle . . . 81
A.3 Atributos del conjunto base en detalle . . . 82
A.4 Importancia de los atributos en el modelo final de RF . . . 84
A.5 Parámetros finales de DNN . . . 85
A.6 Arquitectura Spark . . . 85
A.7 Arquitectura Hadoop . . . 86
A.8 Arquitectura Kafka . . . 86

Lista de acrónimos **87**

Bibliografía **89**

Índice de figuras

1.1 Crecimiento del tráfico móvil global (sin tráfico M2M) hasta 2030. 2
1.2 Relación entre tipos de anomalía de red y amenazas. 4
1.3 Modelo de arquitectura de exportación y recolección de flujos IP. 6
1.4 Esquema de la distribución de tareas de un sistema de distribución heterogéneo. 7
1.5 Funcionamiento de Spark Streaming. 8

2.1 Ciclo de vida de CRISP-DM. 13
2.2 Jerarquía de las fases de CRISP-DM. 13

3.1 Tareas de la fase uno de CRISP-DM: comprensión del negocio. 15
3.2 Función de activación sigmoidea (izquierda) vs. función de activación ReLU (derecha). 19
3.3 Red neuronal sencilla (perceptrón con una capa oculta). 20
3.4 Probabilidad de una clase dados unos valores de sus variables según el Teorema de Bayes 21
3.5 Versión simplificada del Teorema de Bayes bajo la asunción de que las variables son independientes. 21
3.6 Funcionamiento de Random Forest. 23
3.7 Ejemplo de arquitectura compleja de una DNN. 24
3.8 Diagrama de la arquitectura de red en la que se generó el conjunto de datos UNB ISCXIDS201. 26
3.9 Diagrama de Gantt de las tareas del proyecto. 30
3.10 Vista del seguimiento del proyecto. 31

4.1 Tareas de la fase dos de CRISP-DM: comprensión de los datos. 33
4.2 Información general del *dataset* original. A la izquierda, sus atributos originales, a la derecha, la distribución de tipos de flujos. 35
4.3 Gráficas de la distribución de los atributos del Grupo 1 con respecto a *'Target'*. 36

4.4 Gráficas de la distribución de *'source'* con respecto a *'Target'*. 37
4.5 Gráficas de la distribución de *'destination'* con respecto a *'Target'*. 37
4.6 Gráficas de la distribución de *'L2R'* con respecto a *'Target'*. 38
4.7 Mapa de correlaciones positivas mayores a 0.6 entre los atributos modificados de nuestro *dataset*. 40
4.8 Mapa de correlaciones inversas menores a -0.6 entre los atributos modificados de nuestro *dataset*. 41
4.9 Atributos del *dataset* después de eliminar los espurios e inútiles. 42
4.10 Atributos del *dataset* con valores nulos, número de muestras y porcentaje. . . 44

5.1 Tareas de la fase tres de CRISP-DM: preparación de los datos. 45
5.2 Particiones aleatorias que tiene que hacer un árbol de decisión para encontrar un punto atípico (izquierda) vs. particiones necesarias para encontrar un punto normal (derecha). 47
5.3 Particiones aleatorias necesarias generadas por IF durante el entrenamiento. . 48
5.4 Mapa de puntuación de anomalías con dos *clusters* de observaciones normales 48
5.5 Representación de los datos en *'totalSourceBytes'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalSourceBytes'* después de aplicar IF (derecha). 49
5.6 Representación de los datos en *'totalSourcePackets'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalSourcePackets'* después de aplicar IF (derecha). 50
5.7 Representación de los datos en *'totalDestinationBytes'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalDestinationBytes'* después de aplicar IF (derecha). 50
5.8 Representación de los datos en *'totalDestinationPackets'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalDestinationPackets'* después de aplicar IF (derecha). 51
5.9 Circunferencia goniométrica con las horas del día. 53
5.10 Efectos del reescalado en *'totalSourceBytes'*. En (a), la distribución del atributo antes de ser reescalado. En (b), la distribución después. 55
5.11 Efecto de aplicar *random undersampling* al conjunto de datos. 56

6.1 Tareas de la fase cuatro de CRISP-DM: modelado. 59
6.2 Diferencias entre la versión estándar de *cross-validation* (arriba) y la versión estratificada (abajo). 61
6.3 Matriz de confusión para clasificación binaria. 62
6.4 Configuración de la red de búsqueda de parámetros para RF. 64

6.5 Matriz de confusión del modelo RF con mejores parámetros sobre el conjunto de test. . . . 64
6.6 Métricas del rendimiento del modelo RF con mejores parámetros sobre el conjunto de test. . . . 65
6.7 Configuración de la red de búsqueda de parámetros para Bernoulli NB. . . . 65
6.8 Métricas del rendimiento del modelo final Gaussian NB con mejores parámetros sobre el conjunto de test. . . . 65
6.9 Matriz de confusión del modelo final Gaussian NB con mejores parámetros sobre el conjunto de test. . . . 66
6.10 Matriz de confusión del modelo final DNN con mejores parámetros sobre el conjunto de test. . . . 68
6.11 Métricas del rendimiento del modelo final DNN con mejores parámetros sobre el conjunto de test. . . . 68

7.1 Tareas de la fase cinco de CRISP-DM: evaluación. . . . 69

8.1 Tareas de la fase seis de CRISP-DM: despliegue. . . . 71
8.2 Integración de Kafka, Spark Streaming y Hadoop. . . . 73
8.3 Aspecto del panel de control de Zabbix una vez configurado. . . . 76

A.1 Descripción de las variables continuas iniciales del *dataset* UNB ISCX 2012. . . . 81
A.2 Descripción de las variables categóricas iniciales del *dataset* UNB ISCX 2012. . . . 82
A.3 Descripción de las variables continuas finales empleadas para entrenar los modelos. . . . 82
A.4 Descripción de las variables categóricas finales empleadas para entrenar los modelos. . . . 83
A.5 Importancia que le da el modelo final de RF a cada atributo. . . . 84
A.6 Arquitectura de Spark [1]. . . . 85
A.7 Arquitectura de Hadoop [2]. . . . 86
A.8 Arquitectura de Kafka [3]. . . . 86

Índice de tablas

3.1 Coste de cada tarea según la estimación salarial del investigador. 31
3.2 Coste total del proyecto según las estimaciones salariales de los dos perfiles. . 31

7.1 Métricas de cada modelo dando por hecho que la clase positiva es ser un flujo de ataque. 70

8.1 Especificaciones de las máquinas virtuales levantadas para el despliegue. . . . 74

A.1 Parámetros finales de DNN. 85

Capítulo 1

Introducción

El número de máquinas que hay hoy en día conectadas a Internet, compartiendo su información a través de nuestras redes, es ingente. Para hacernos una idea, en el momento en el que se escribe este texto hay 97.528 GB/s de tráfico cruzando Internet [4]. Además, lejos de estabilizarse, la estimaciones al respecto parecen coincidir en que la tendencia es que la cantidad siga aumentando.

Si algo se ha hecho evidente en estos últimos meses de reajuste acorde a la situación global actual, marcada por la aparición y avance del SARS-CoV-2, es que las redes ya no solo son una herramienta necesaria para alcanzar estándares de competitividad en nuestra producción regular. Además, también conforman el último resorte con el que contamos para que el mundo no se detenga, cuando nos vemos obligados a cambiar la manera en la que la sociedad se organiza [5].

En la figura 1.1 podemos ver una estimación de la Unión Internacional de Telecomunicaciones (UIT) sobre el futuro de las Telecomunicaciones Móviles Internacionales (IMT) para 2020 y en adelante. Como se puede observar, sólo el tráfico de dispositivos móviles, y sin incluir tráfico M2M, podría crecer a un ratio anual de hasta un 54%, llegando a suponer 4.394 EB de tráfico global por mes en 2030 [6].

En vista de lo anterior, cabe preguntarnos si los mecanismos a los que confiamos la protección de nuestras redes son adecuados, y si lo seguirán siendo en el futuro. Hasta ahora, hemos venido empleando predominantemente métodos basados en políticas y reglas para protegerlas. Sin embargo, a medida que aumenta el número y variedad de dispositivos, estas técnicas se quedan cortas en velocidad de procesamiento y adaptabilidad. Por suerte, es justo en esas condiciones en las que las técnicas de *machine learning*, a partir de ahora ML, muestran sus puntos fuertes y se posicionan como una opción fuerte para salvaguardar los titanes en los que nuestras redes se han convertido.

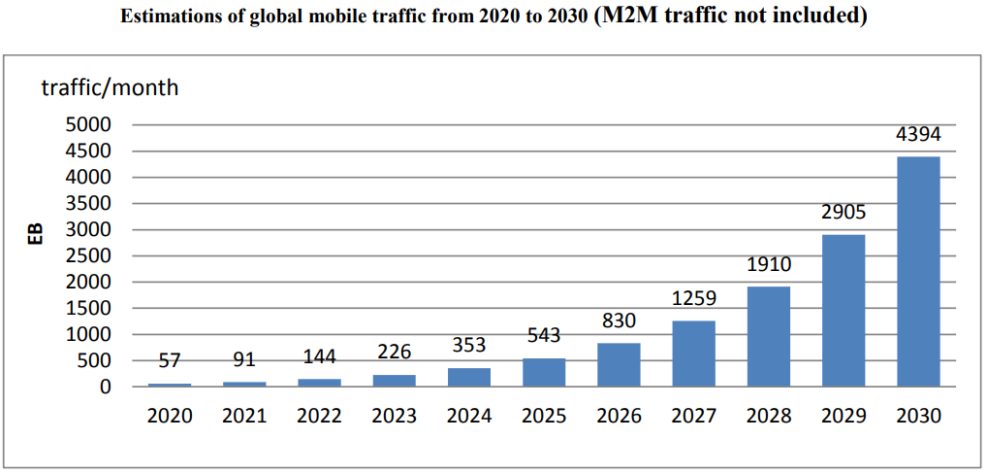


Figura 1.1: Crecimiento del tráfico móvil global (sin tráfico M2M) hasta 2030.

1.1 Contextualización

1.1.1 Anomalías, amenazas, y cómo detectarlas

Se consideran anomalías a los patrones en nuestros datos que no se adaptan a las características definidas como el comportamiento habitual del conjunto. La detección de estas es una tarea útil para la detección de intrusiones (ID), ya que entre las causas principales que pueden provocar su aparición están los ciberataques. Un aspecto importante relacionado con las anomalías es su naturaleza. Podemos clasificarlas como [7]:

- Anomalía de punto: cuando una instancia particular de los datos se desvía del patrón general del conjunto. Un ejemplo sería que el consumo de gasolina de una persona pasase de 5 a 10 litros en un día aleatorio.
- Anomalía de contexto: cuando una instancia de los datos se comporta diferente en un contexto en particular. Un ejemplo sería el aumento en el gasto de una tarjeta de crédito en un periodo festivo.
- Anomalía colectiva: cuando una colección de instancias de datos similares se comporta de manera anómala con respecto al conjunto de datos entero. Un ejemplo sería un valor bajo prolongado en un electrocardiograma.

Por otro lado, empleando una definición ligeramente simplista de los ataques informáticos que buscamos detectar, como cualquier acción perjudicial cuyo objetivo sea comprometer una red, podemos dividirlos en la siguiente tipología [7]:

- Denegación de Servicio o *Denial of Service* (DoS): consiste en el abuso de los derechos de acceso a recursos de una red o host con el objetivo de mermar la disponibilidad del servicio.
- *Probe*: también se conocen como ataques de reconocimiento, y son una forma común de conseguir información sobre los tipos y números de máquinas conectadas a una red, así como sobre el tipo de software y aplicaciones instaladas en un host. Un ataque *Probe* suele ser el primer paso en el intento de comprometer un sistema, aunque el propio ataque no esté causando daño.
- *User to Root* (U2R): se lanzan cuando se pretende ganar acceso ilegítimo a una cuenta administrativa para manipular o abusar de recursos importantes. Para llevarlo a cabo, primero se suele emplear ingeniería social o sniffing para conseguir una cuenta normal de usuario, y a continuación exploits o vulnerabilidades que permitan escalar privilegios.
- *Remote to User* o *Remote to Local* (R2U o R2L): se dan cuando un atacante quiere ganar acceso local como usuario de una máquina objetivo, para obtener el privilegio de enviar paquetes en la red en la que se encuentra esa máquina. A menudo el atacante emplea un método de ensayo y error para adivinar la contraseña mediante scripts automatizados, métodos de fuerza bruta, etc.

En base a las descripciones anteriores de los tipos de anomalía y de ataque, es posible identificar relaciones entre ambos conjuntos. Según Mohiuddin Ahmed et al. [7], las características de los ataques DoS se corresponderían con una anomalía colectiva, ya que podemos considerar las numerosas peticiones de conexión necesarias para perpetrarlo como una colección de instancias de datos que se comportan de manera anómala. Los ataques *Probe* se podrían asociar a las anomalías de contexto y, por último, los ataques U2R y R2L, al depender de ciertas condiciones específicas y ser más difíciles de llevar a cabo, se clasificarían como anomalías de punto. En la figura 1.2 [7] se puede observar la asociación que acabamos de definir.

Una vez entendida la relación que hay entre anomalías y ataques, podemos empezar a plantearnos cómo detectar las anomalías que, en efecto, representan amenazas para nuestras redes. Las principales alternativas con las que contamos para crear sistemas de detección de intrusiones en red son: detección estadística de anomalías, detección de anomalías basada en teoría de la información, detección de anomalías basada en *clustering*, y detección de anomalías basada en clasificación, siendo esta última la vía que exploraremos a lo largo de este trabajo [7].

Asimismo, existen tres formas de actuar en la detección de anomalías basada en clasificación. La primera se basa en reconocer firmas previamente introducidas en el sistema para

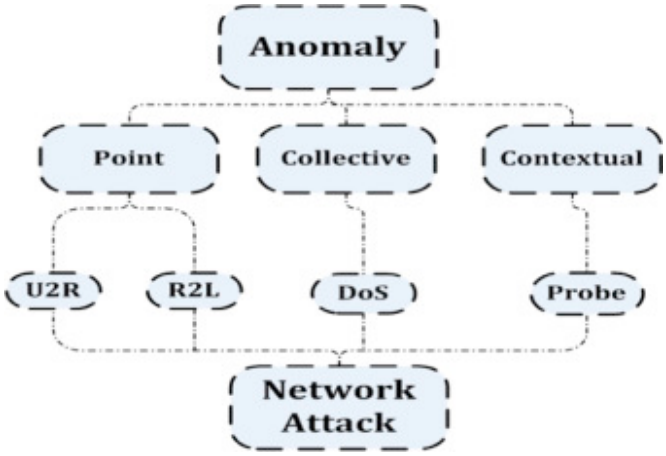


Figura 1.2: Relación entre tipos de anomalía de red y amenazas.

detectar ataques ya conocidos. Es efectiva detectando ataques registrados, pero no puede detectar nuevas amenazas. Para superar este problema, la segunda forma, detección basada en anomalías, construye su conocimiento entorno al concepto de perfil de tráfico normal, considerándose como anomalía el tráfico que se desvíe del perfil establecido. Esta opción es efectiva a la hora de detectar amenazas nuevas, pero su excesiva dependencia de tener un perfil de tráfico normal preciso y actualizado la hace tender a producir falsos positivos. Finalmente, existe un término medio, en el que intentamos compensar las carencias de las anteriores formas combinándolas. En este trabajo se ha optado por la opción híbrida [7] [8].

1.1.2 *Machine learning* en la detección de intrusiones

La potencia de las técnicas de ML radica fundamentalmente en su capacidad de encontrar patrones en conjuntos de datos de forma autónoma. Es la evolución lógica de las condiciones en programación, y la única opción que conocemos para cuando el problema que abarcamos es demasiado complejo como para ser perfilado manualmente.

Por otro lado, la situación que acabamos de describir es extrapolable a la que existe actualmente en nuestras redes. Empiezan a ser demasiado complejas como para ser perfiladas manualmente, y los sistemas convencionales sólo serán viables por sí solos cierto tiempo. Cuando llegue ese momento, es necesario que ya contemos con alternativas que nos ayuden a conducir nuestra seguridad hacia un paradigma más escalable y adaptable. Uno de los caminos para hacerlo es aplicando ML a nuestro dominio.

Las técnicas de ML llevan años siendo estudiadas en materia de detección de intrusiones en red. Por lo que sabemos, podrían solucionar parte de los problemas que tenemos con los sistemas convencionales, y facilitar el camino para solucionar los demás. Esto no significa que su aplicación sea trivial, o que no haya retos que superar, pero sí que deberíamos seguir

enriqueciendo la investigación al respecto, de manera que aplicarlas a entornos reales sea cada vez una opción más viable.

1.1.3 *Big Data* y *machine learning*

Cogiendo prestada la definición de Gartner [9], cuando hablamos de *Big Data* nos referimos a activos de información con un gran volumen, velocidad y/o variedad, que requieren formas de procesamiento nuevas y rentables que permitan una percepción mejorada, toma de decisiones, y automatización de procesos.

Las técnicas de ML no son, como cabe esperar, indiferentes a la extensión de los datos cuando se trata del entrenamiento, o de la capacidad de predicción en tiempo real. Sin embargo, son propensas a ser escalables, dándonos margen de maniobra en su capacidad de procesamiento si decidimos conjugarlas con técnicas de manejo de *Big Data*. A continuación, detallaremos los mecanismos empleados en nuestro proyecto para hacer viable computacionalmente nuestra tarea: flujos de red, computación distribuida, y computación distribuida basada en flujos.

Flujos de red

Los sistemas de detección de intrusiones en red tradicionales se basaban en la inspección pormenorizada del contenido (*payload*) de cada paquete, una técnica que se conoce como *Deep Packet Inspection*, de ahora en adelante DPI. Sin embargo, esta técnica plantea un cuello de botella por el alto coste que supone en términos de procesamiento, lo que dificulta su aplicación en redes grandes [10].

Un flujo IP de red es un grupo de paquetes IP con características en común, pasando un punto de monitorización en un intervalo de tiempo específico. La monitorización basada en flujos tiene su fundamento en la información de las cabeceras de los paquetes. Con este acercamiento, pasamos de analizar los contenidos individuales de los paquetes, a los patrones de comunicación en nuestra red, y en consecuencia, conseguimos disminuir la carga computacional necesaria. Por poner un ejemplo, en el experimento de Anna Sperotto et al. [10] sobre la red de la Universidad de Twente, se calculó que el ratio entre paquetes exportados con flujos, y los paquetes en la red era de media un 0.1%. Por otro lado, los datos basados en flujo son más compatibles con requerimientos de privacidad que DPI por no depender del *payload*.

Finalmente, cabe destacar que el agrupamiento del tráfico en flujos no sólo es una herramienta útil para la reducción de la extensión de los datos, sino que también es adecuada para facilitar la exportación del tráfico a colecciones remotas. Esto propicia el desarrollo de sistemas basados en colecciones de datos distribuidas [10].

En la figura 1.3 [10] podemos observar un ejemplo de arquitectura de exportación y recolección de flujos IP.

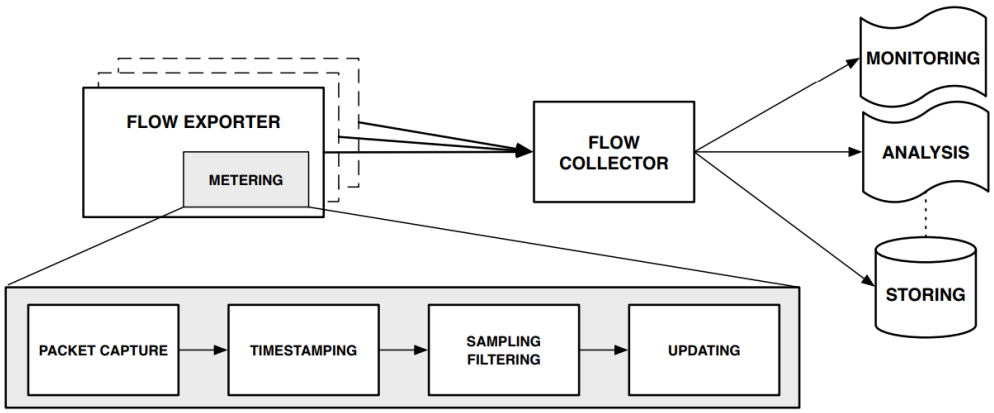


Figura 1.3: Modelo de arquitectura de exportación y recolección de flujos IP.

Computación distribuida

Citando de nuevo a Gartner [11], en la computación distribuida los datos y aplicaciones están divididos entre máquinas o sistemas diferentes, pero conectados e integrados mediante servicios de red y estándares de interoperabilidad, de tal manera que funcionan como un único entorno. Las características clave de los sistemas de computación distribuida son [12] [13]:

- Heterogeneidad: al establecer un sistema de comunicación por mensajes, es posible emplear tecnología diferente en los nodos de nuestros sistemas de computación distribuida. Concretamente, podemos tener variedad de redes, hardware, sistemas operativos, lenguajes de programación e implementaciones.

- Ampliable: esta propiedad hace referencia a la capacidad de un sistema para ser extendido y reimplementado. El sistema distribuido tiene que ser ampliable en *hardware* y *software*. Para hacerlo posible, es necesario contar con una interfaz detallada y bien definida de componentes, estandarizar las interfaces de los mismos y hacer que los nuevos componentes sean fácilmente integrables con los ya existentes.

- Seguridad: es posible que los conjuntos de datos que se ponen a disposición de los sistemas distribuidos tengan un alto valor intrínseco para sus usuarios. La seguridad de los mismos es, por lo tanto, de considerable importancia. Hay tres componentes de seguridad que asegurar: confidencialidad, integridad y disponibilidad.

- Escalabilidad: esta propiedad hace referencia a cómo de bien el sistema se comporta cuando tiene que lidiar con un número de usuarios significativamente mayor. Para mantener el sistema efectivo, compensamos el aumento de carga con más nodos.

- Tolerancia a fallos: cualquier componente en un sistema distribuido puede fallar. Pese a todo, es necesario que tenga un diseño que permita, en la medida de lo posible, que el sistema se mantenga disponible después de que haya tenido lugar cualquier incidente con sus componentes.

- Concurrencia: la concurrencia de un sistema hace referencia a su capacidad de que varias tareas se ejecuten en el mismo momento. La ejecución concurrente de actividades se lleva a cabo en diferentes componentes, corriendo en distintas máquinas, como parte del sistema distribuido. La concurrencia disminuye la latencia e incrementa el rendimiento del sistema distribuido.

- Transparencia: los usuarios y desarrolladores deberían tener una percepción de los sistemas distribuidos como un todo en vez de como una colección de elementos que cooperan. La transparencia puede ser de varios tipos como acceso, escalado, movilidad, etc.

Podemos observar en el diagrama de la figura 1.4 [14] el modelo de un sistema de distribución heterogéneo.

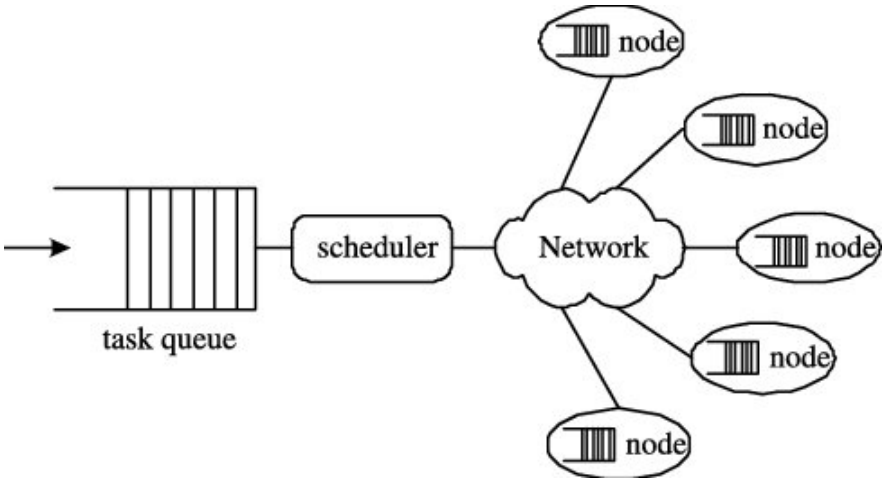


Figura 1.4: Esquema de la distribución de tareas de un sistema de distribución heterogéneo.

Computación distribuida basada en flujos

El agrupamiento del tráfico de red en flujos es, como ya se ha explicado, una técnica conveniente por varias razones. No obstante, en lo que respecta a los objetivos de este trabajo, dicho agrupamiento también tiene una desventaja: nos impide realizar una detección de intrusiones completamente en tiempo real, ya que la creación de los flujos no es inmediata. De todas formas, sí es posible acercarnos empleando técnicas de computación distribuida.

En el pasado, la solución con más acogimiento para el procesamiento de *Big Data* era el paradigma MapReduce, que desde su concepción ha hecho el procesamiento de complejos y extensos volúmenes de datos, fácil y eficiente. MapReduce está principalmente pensado para procesamiento en lotes, en el que almacenamos los datos de entrada durante un lapso de tiempo para después procesarlos. Este paradigma, aunque útil en ciertos dominios, en principio no permite implementar un sistema de ML en línea que procese continuamente nuevos flujos, y detecte intrusiones en la red. Para esto último, tenemos que cambiarnos al paradigma del *stream processing*, que nos permite trabajar en contextos en los que los datos de entrada han de ser procesados sin ser totalmente guardados, y una *query* tiene que mantenerse activa de manera continua [15].

Actualmente no hay demasiadas soluciones diseñadas para trabajar con *streams* puros de *Big Data*. No obstante, podemos emular esta forma de procesamiento con el *micro-batching*. El *micro-batching* representa una adaptación del *paradigma* MapReduce al *stream processing*. Consiste en tratar el *stream* como una secuencia de pequeños lotes, y entregárselos al sistema de procesamiento en cortos intervalos de tiempo [15]. Esta es la solución que nos propone Spark Streaming, y cuyo diagrama podemos observar en la figura 1.5 [16].

Para acercarnos aún más al verdadero procesado en *streaming*, podemos usar Spark Structured Streaming. La diferencia principal con Spark Streaming es se hace transparente el concepto de *micro-batch* para el usuario. Structured Streaming nos permite trabajar como si estuviéramos usando lotes de datos estáticos, mientras el motor Spark SQL se encarga de que se ejecute de manera incremental y continua según llegan los datos, en vez de cada cierto tiempo. Este es el acercamiento que emplearemos.

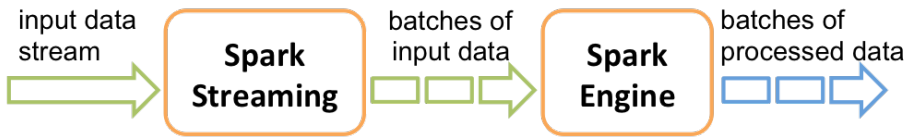


Figura 1.5: Funcionamiento de Spark Streaming.

1.2 Objetivos

Este trabajo tiene dos objetivos principales. Por un lado, la comprensión y comparación de diferentes algoritmos de aprendizaje máquina empleados en la clasificación de tráfico de red, y por el otro, el estudio de mecanismos de distribución para los mismos. En aras de satisfacerlos, realizaremos una prueba de concepto consistente en la implementación de un sistema de detección de intrusiones en red. Concretamente, afrontaremos los siguientes puntos:

- Seleccionar algoritmos candidatos de ML: se tendrá en cuenta tanto que hayan obtenido buenos resultados en proyectos anteriores de dominios similares, como que sean diferentes entre sí. Se busca con ello enriquecer la comparativa.
- Seleccionar conjunto de estudio y aplicar ingeniería de características en sus datos: se elegirá un conjunto de datos apropiado para el dominio, y se trabajará sobre el mismo con la pretensión de representar los datos de tal manera que los algoritmos candidatos puedan aprovecharse de ellos.
- Configurar e implementar los algoritmos seleccionados: se realizará una configuración atendiendo a sus características individuales, y los parámetros relevantes que haya disponibles. Después se implementarán modelos basados en los algoritmos configurados, para poder evaluar su efectividad y mejorar la configuración inicial.
- Seleccionar y evaluar tecnologías de distribución: se buscarán alternativas en materia de despliegue distribuido de aplicaciones, que sean capaces de dar soporte a modelos de ML. También se valorarán herramientas de monitorización de sistemas distribuidos.

1.3 Estructura del proyecto

En lo que respecta a la distribución de este documento, se ordenarán los contenidos de la siguiente manera:

- **Capítulo 2. Metodología.**
  - Explicación de la metodología empleada para la realización del proyecto.
- **Capítulo 3. Comprensión del negocio.**
  - Exposición de los conceptos relevantes para el proyecto.
  - Evaluación y selección de herramientas para llevar a cabo el proyecto.
  - Planificación del proyecto.
- **Capítulo 4. Comprensión de los datos.**
  - Exposición de los mecanismos empleados para la descripción, exploración y verificación de la calidad de los datos.
- **Capítulo 5. Preparación de los datos.**
  - Exposición de los mecanismos que se hayan empleado para la selección, limpieza, transformación e integración de los datos según cada algoritmo de ML y sus asunciones de modelado.

- **Capítulo 6. Modelado.**
  - Explicación en detalle del proceso de entrenamiento, hiperparametrización y validación de las técnicas de ML empleadas.
- **Capítulo 7. Evaluación.**
  - Exposición de los mecanismos empleados para la evaluación de cada una de las partes del proyecto.
- **Capítulo 8. Despliegue.**
  - Exposición de los aspectos relevantes sobre la creación del sistema distribuido que da soporte al algoritmo de ML seleccionado.
  - Explicación de cómo se aborda la monitorización del sistema, y revisión de los resultados obtenidos.
- **Capítulo 9. Conclusiones y líneas futuras.**
  - Balance de los resultados con respecto a los objetivos establecidos, y declaración de lecciones aprendidas.
  - Indicación de futuros desarrollos posibles en base a resultados, y observaciones realizadas a lo largo del proyecto.

Capítulo 2

Metodología

La extracción de conocimiento a partir de nuestros datos debería llevarse a cabo mediante procesos y procedimientos sistemáticos, a través de pasos bien definidos. Una metodología es una guía general de las técnicas y actividades que debemos emplear dentro de un dominio específico. Las metodologías no se basan en tecnologías o herramientas concretas, sino que nos aportan un marco de trabajo para adquirir resultados mediante el empleo de un amplio rango de métodos, procesos y heurísticas [17].

En lo referente a la aplicación de técnicas de ciencia de datos o *data science* para la ciberseguridad, hay varias metodologías disponibles. Entre estas se pueden destacar SEMMA *(Sampling, Exploring, Modifying, Modeling, and Assessing)*, KDD *(Knowledge Discovery in Databases)* y sobre todo CRISP-DM *(Cross Industry Standard Process for Data Mining)*, que en los últimos años se ha mantenido a la cabeza, habiendo documentos que le atribuyen hasta un 42% de implantación en 2014 [17].

Pese al gran acogimiento que ha tenido y aún tiene CRISP-DM, hoy en día ya se conocen algunas limitaciones de esta metodología. Ya existen alternativas que solucionan estas carencias, pero para este proyecto hemos decidido continuar con CRISP-DM. Entre las razones principales, consideramos que el proyecto se beneficiaría de una metodología extensamente implantada y entendida, más que de alternativas potencialmente más adaptables, pero también más experimentales y menos estudiadas.

2.1 CRISP-DM

Cross Industry Starndard Process for Data Mining nació en 1996 a partir de una iniciativa de las empresas NCR, Daimler y OHRA; siendo desarrollada por SPSS y Teradate. Está muy bien documentada y cuenta con numerosos casos de uso disponibles para consultar. Además, después de tanto tiempo sigue siendo útil en todo tipo de proyectos, aunque para ciertos ámbitos necesite ajustes.

Entre las razones de su éxito se encuentra la independencia entre el proceso que propone CRISP-DM y las herramientas de minería de datos que se decidan emplear, y que sus pasos son tan lógicos que parecen casi de sentido común. De hecho, existen numerosas metodologías más recientes que se basan en CRISP-DM e intentan mejorar sus puntos débiles.

CRISP-DM consta de cuatro niveles de abstracción, organizados de forma descendente de más general a más específico. En el nivel superior de la jerarquía encontramos las seis fases principales, que interactúan entre ellas de manera iterativa a lo largo del proyecto. Podemos ver estas fases en la figura 2.1 [17]. A continuación se indica qué parte del proyecto abarca cada una de esas seis partes:

1. Comprensión del negocio o *business understanding*: en esta fase se busca entender el problema, o los requisitos y objetivos del proyecto, desde la perspectiva del negocio, en este caso una aplicación de ciberseguridad. A continuación, se transformará esta percepción en un esbozo preliminar del problema de minería de datos en el que se traduce.

2. Comprensión de los datos o *data understanding*: al comienzo de esta fase partimos de una colección de datos, y a partir de ella, realizamos tareas con el objetivo de entenderlo, encontrar sus problemas de calidad, o incluso identificar tendencias para desarrollar hipótesis tempranas sobre su información subyacente. Normalmente al final de esta fase acabamos con un conjunto de datos modificado.

3. Preparación de los datos o *data preparation*: esta fase engloba todas las actividades que sean necesarias para construir el conjunto final a partir de los datos de los que partimos de fases anteriores.

4. Modelado o *modeling*: en esta fase se aplican técnicas y estrategias de modelado. Los parámetros específicos y prerrequisitos de los modelos deberían ser identificados y calibrados según el tipo de datos que se introduzca para obtener valores óptimos.

5. Evaluación o *evaluation*: en este punto, el modelo o modelos obtenidos serán puestos a prueba para medir su eficacia, y lo bien que son capaces de generalizar sobre conjuntos nuevos de datos. Según los resultados obtenidos sabremos si hemos alcanzado los objetivos clave. El resultado es el conjunto de los modelos válidos.

6. Despliegue o *deployment*: esta fase consiste en desplegar una representación en código del modelo o modelos finales, con el objetivo de evaluar o categorizar nuevos datos. La nueva información a la que expongamos nuestros modelos ha de estar organizada, estructurada y presente de manera que pueda ser utilizada. Esto incluye toda la preparación de los datos y cualquier otro paso necesario para convertirlos en una entrada adecuada para los modelos.

Estas fases están a su vez estructuradas en actividades de segundo nivel. Las tareas generales del segundo nivel se proyectan en tareas específicas del tercero, donde se describen las acciones que deben desarrollarse para situaciones concretas. Por ejemplo, si en el segundo nivel tenemos la tarea general "Construcción del modelo", dentro de la fase "Modelado", en el tercer nivel se han de indicar las tareas que tienen que desarrollarse para un caso específico, como puede ser "Configurar parámetros". El cuarto nivel está conformado por el conjunto de acciones, decisiones y resultados sobre el proyecto específico [18]. En la figura 2.2 [18] se puede ver con más claridad la jerarquía de fases.

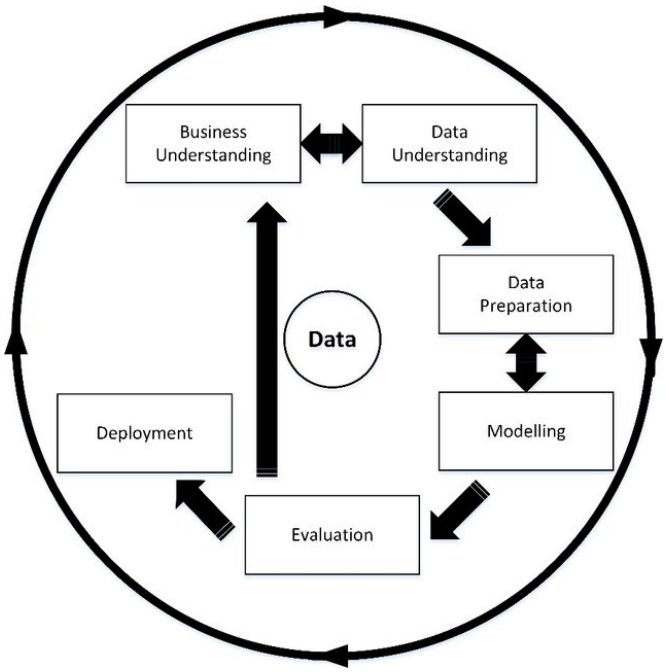


Figura 2.1: Ciclo de vida de CRISP-DM.

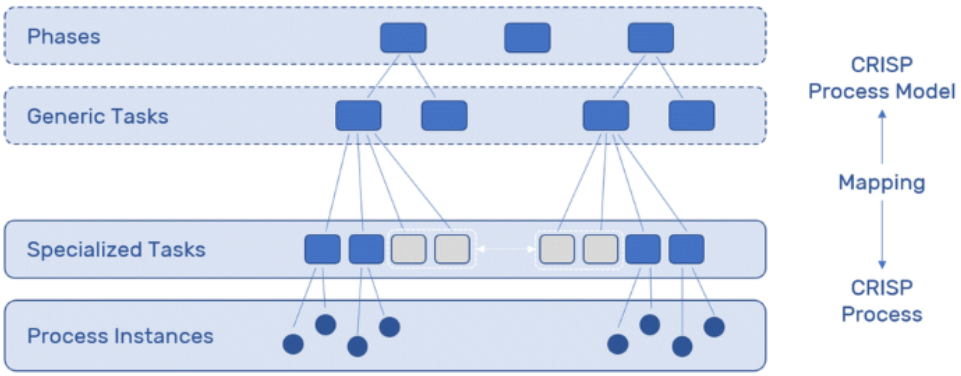


Figura 2.2: Jerarquía de las fases de CRISP-DM.

Los siguientes seis capítulos de esta memoria están dedicados a cada una de las seis fases primarias de la metodología. En ellos especificaremos, de manera sintetizada, las actividades llevadas a cabo para cumplir con los pasos sugeridos por CRISP-DM en cada fase, a lo largo de las diferentes iteraciones realizadas.

Capítulo 3

Comprensión del negocio

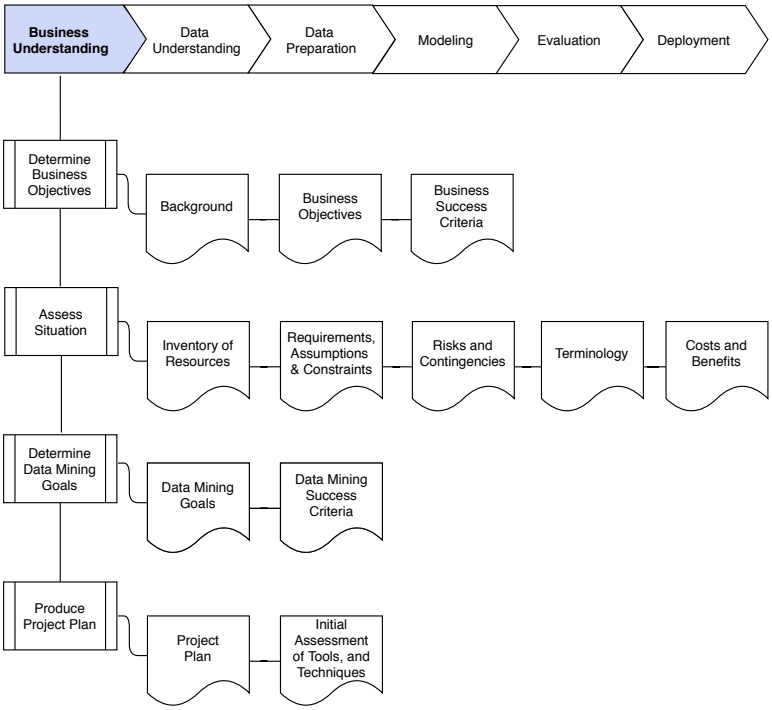


Figura 3.1: Tareas de la fase uno de CRISP-DM: comprensión del negocio.

En este capítulo hablaremos de la primera fase del ciclo de vida CRISP-DM en el marco específico de nuestro proyecto. Como se puede observar en la figura 3.1, las actividades que se nos proponen para esta fase son:

- Determinación de los objetivos del negocio: el propósito fundamental de esta tarea es definir los objetivos del proyecto desde el punto de vista del negocio. En nuestro caso, ya quedaron especificados en la sección 1.2.
- Evaluación de la situación: el fin de esta tarea es adquirir una visión detallada sobre los

recursos, restricciones, asunciones, y en general, todos los factores relevantes que han de ser considerados para tanto la realización de nuestros objetivos, como la planificación del proyecto. Esta tarea es abordada en el capítulo actual, concretamente en las secciones 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 y 3.7.

- Determinación de las metas a alcanzar: para la realización de esta tarea debemos responder a la pregunta de cómo pueden ayudarnos las técnicas de ML a conseguir los objetivos de nuestro proyecto. En nuestro caso, la respuesta es mediante la detección de amenazas en las redes.

- Producción de un plan de proyecto: esta tarea consiste en la creación un plan para alcanzar los objetivos del proyecto. Este deberá especificar los pasos a seguir. El plan será detallado también en este capítulo, concretamente en la sección 3.8.

3.1 *Machine learning*

Para que una máquina solucione un problema, necesitamos un algoritmo. Sin embargo, hay algunas tareas para las que, si bien tenemos claro cuáles son la entrada y la salida que queremos, no somos capaces de encontrar una serie de pasos que consigan la transformación. Un ejemplo de este tipo de tareas podría ser detectar si un correo electrónico es deseado o no. Esto se debe a que las características de lo que consideramos correo no deseado cambian a lo largo del tiempo [19].

Para compensar el no saber los pasos necesarios para una transformación, disponemos de una extensa cantidad de ejemplos. Lo ideal sería, por lo tanto, que la máquina observase los ejemplos y encontrase una aproximación que le permitiera, como en nuestro caso, saber que un correo es no deseado, aunque no estemos seguros de cuáles son todos los factores envueltos en nuestra decisión. Este es el nicho del ML. Cuando empleamos estas técnicas, los patrones en los datos son la llave que nos ayuda a entender el proceso subyacente, o a hacer predicciones sobre datos aún no catalogados, garantizando hasta cierto punto que van a ser predicciones correctas [19].

La aplicación del ML a grandes volúmenes de datos se denomina minería de datos o *data mining*. Estas técnicas se consideran parte de la inteligencia artificial. En líneas generales, para que un sistema sea considerado inteligente, tiene que ser capaz de aprender en un entorno cambiante, de manera que el diseñador no tendrá que proveerlo de todas las posibles soluciones de antemano [19].

3.1.1 Clasificación de técnicas

Dependiendo de las características del conjunto de datos que queramos emplear para modelar, deberíamos plantearnos aplicar algoritmos que usen un tipo de aprendizaje u otro. Los diferentes tipos son:

- Aprendizaje supervisado: las observaciones de nuestro conjunto de datos cuentan con etiquetas que nos permiten saber cuál es la clase a la que pertenecen. Estos ejemplos etiquetados permiten definir un criterio de evaluación del sistema. Por otro lado, los algoritmos de esta familia pueden dividirse según el tipo de tarea que se aborda. Los principales son:
  - Clasificación: la empleamos cuando el objetivo es predecir la etiqueta de clase, es decir, un valor categórico. A veces se separa en clasificación binaria, cuando la predicción es sólo entre dos clases, y clasificación multiclase, cuando el espacio de clases es mayor. Un ejemplo de clasificación binaria relacionado con la ciberseguridad sería la clasificación de flujos de red como pertenecientes a un ataque.
  - Regresión: para las tareas de este tipo, la variable objetivo es un número continuo. Una manera de aplicar las tareas de regresión en el ámbito de la ciberseguridad sería calculando la probabilidad de que una actividad sea fraudulenta, dado un conjunto de datos con variables relevantes.
- Aprendizaje no supervisado: recurrimos a este aprendizaje cuando no contamos con ninguna etiqueta que indique la clase de las observaciones. En este tipo no se busca predecir la salida correcta, sino hacer deducciones a partir de los patrones que se puedan encontrar en los datos. Esta familia de algoritmos diverge en dos tipos principales según la tarea a abarcar:
  - Agrupamiento o *clustering*: es similar a la clasificación, siendo la mayor diferencia que no se conoce la clase de las observaciones. Uno de los campos de la ciberseguridad para los que se cree que este tipo de algoritmos es adecuado, es el análisis forense.
  - Asociación: se emplea cuando queremos descubrir reglas de asociación, como por ejemplo cuántas personas que compran un producto también compran otro. En ciberseguridad, se le ha encontrado utilidad a estos algoritmos para respuesta a incidentes y gestión de riesgos.
- Aprendizaje semi-supervisado: se usan técnicas tanto de aprendizaje supervisado como no supervisado. Es decir, contamos con observaciones etiquetadas y no etiquetadas para el entrenamiento.

- Aprendizaje por refuerzo: consiste en usar errores estimados como premios o penalizaciones. Si el error es grande, la penalización es alta y el premio bajo. Si el error es bajo, la penalización es baja y el premio alto. Se busca por ende que el sistema maximice su noción de recompensa. Las características más relevantes de este tipo de aprendizaje son la búsqueda por ensayo y error, y el refuerzo diferido.

3.1.2 *Deep Learning*

Deep Learning, de aquí en adelante DL, es una rama del ML que destaca por su capacidad de generalización cuando la naturaleza de los datos con los que se trabaja es compleja, y alberga una alta dimensionalidad. Además, DL permite entrenamiento escalable con conjuntos de datos extensos. Estas características son relevantes en el ámbito de la detección de intrusiones en red, no sólo por el exacerbado volumen de datos, sino porque los modelos que se generen para este dominio han de ser capaces de generalizar lo suficiente como para detectar nuevos patrones de ataque [20].

Las técnicas de DL estuvieron inspiradas en sus comienzos por modelos computacionales del cerebro humano, lo que propició la popularización del uso de nombres como redes de neuronas artificiales, o ANNs, por sus siglas en inglés. Sin embargo, pese a que el punto de vista neurocientífico fue en su día un referente adecuado, actualmente esta rama del ML sigue su propio camino [20].

La razón por la que el DL lleva tomando más y más fuerza desde su resurgimiento en los 2000, es una combinación del incremento tanto en la capacidad de computación, como en los datos disponibles. El DL es más eficaz cuando tiene un gran volumen de datos del que aprender. Concretamente, la investigación al respecto parece indicar que para alcanzar buenos resultados, es necesario contar con al menos 5.000 observaciones por categoría [20].

Redes neuronales

Una red neuronal artificial es un sistema de procesamiento de información. Cuenta con ciertas características similares a las que podemos observar en las redes neuronales biológicas. Además, se compone de unidades de computación simples llamadas nodos o neuronas. Todas las neuronas del sistema están conectadas con alguna otra. Estas conexiones se efectúan a través de enlaces de comunicación directa, o sinapsis, y cada sinapsis tiene un peso asociado. Hay diferentes tipos de redes neuronales, y pueden ser categorizadas según su [20]:

- Arquitectura: haciendo referencia al patrón en el que las conexiones entre neuronas están dispuestas.
- Algoritmo de aprendizaje: haciendo referencia a la manera en la que los pesos de las sinapsis son calculados.

- Función o funciones de activación: haciendo referencia a las funciones empleadas por las capas individuales de la red neuronal.

Cada neurona mantiene un estado interno propio, que es determinado por la función de activación aplicada a sus entradas. Una vez activadas, las neuronas envían el resultado a todas las otras con las que están directamente conectadas, para continuar con el ciclo en la siguiente capa de la red. Dos de las funciones de activación más habituales para una neurona son la función sigmoidea y la función rectificador, o ReLU por sus siglas en inglés. Podemos observar ambas funciones en la figura 3.2 [20].

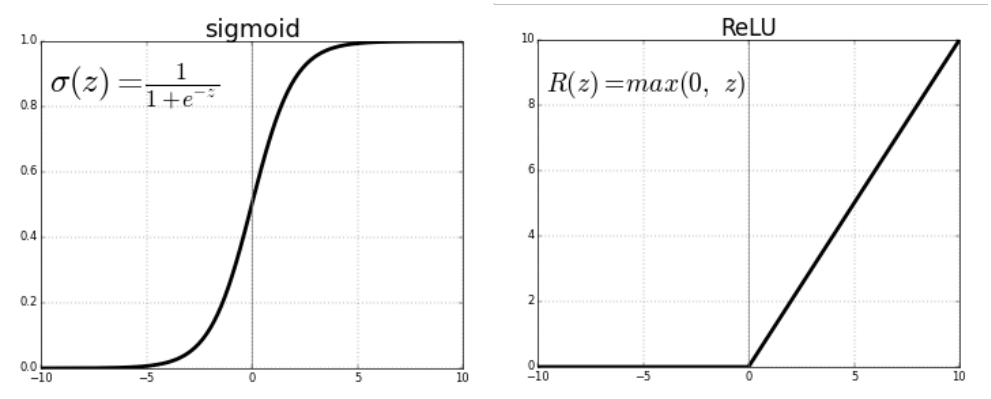


Figura 3.2: Función de activación sigmoidea (izquierda) vs. función de activación ReLU (derecha).

Por otro lado, en la figura 3.3 podemos ver un ejemplo de red neuronal simple. Más específicamente, se trata de un perceptrón de una sola capa oculta. En este ejemplo, la neurona Y recibe las entradas x_1, x_2 y x_3. Cada una de las tres conexiones entre las entradas y la neurona Y están representadas por variables con los pesos $w_1^{[1]}$, $w_2^{[1]}$, y $w_3^{[1]}$. Por consiguiente, la entrada y_{in} a la neurona Y se calcula como la suma de las tres señales ponderadas de entrada x_1, x_2, y x_3 [20]:

$$y_{in} = w_1^{[1]}x_1 + w_2^{[1]}x_2 + w_3^{[1]}x_3$$

Esto se generaliza a un número de entradas n como:

$$y_{in} = \sum_{i=1}^{n} x_i w_i^{[l]}$$

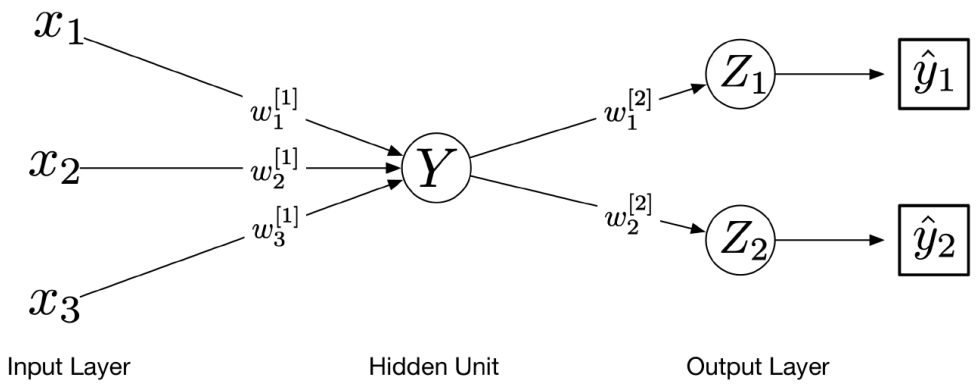


Figura 3.3: Red neuronal sencilla (perceptrón con una capa oculta).

Después de que la neurona Y aplique su función de activación en la entrada, obtendrá una salida y_{out}. En este ejemplo, la neurona Y está conectada a las neuronas de la capa de salida Z_1 y Z_2, con pesos $w_1^{[2]}$ y $w_2^{[2]}$ respectivamente. Por lo tanto, Y mandará y_{out} tanto a Z_1 como a Z_2. Sin embargo, los valores que estas dos recibirán como input variará, ya que cada una de las conexiones hacia estas tiene un peso diferente asociado [20].

En la capa de salida, Z_1 y Z_2 tienen su propia función de activación, que genera la salida de la red, $\hat{y_1}$ e $\hat{y_2}$, respectivamente. En clasificación, estas podrían ser las probabilidades respectivas de que una entrada determinada X sea de la clase 0 o 1 [20].

En cada paso del entrenamiento, las redes neuronales ajustan los pesos de sus conexiones, en un esfuerzo por minimizar la función de pérdida. Actualizar pesos es la manera principal que tiene la red neuronal de aprender. Para hacer este ajuste de pesos la red emplea un algoritmo llamado retropropagación o *backpropagation*. Esta técnica hace uso de la funciones de optimización en los pesos de las conexiones de las neuronas, para minimizar el error en la salida generada por la red [20].

3.2 Algoritmos

En la selección de algoritmos para este proyecto se tuvieron en cuenta, principalmente, dos aspectos: por un lado, que hubieran sido aplicados anteriormente en un dominio similar con resultados positivos [21]; por el otro, buscamos algoritmos lo más divergentes entre sí que fuera posible, con el objetivo de enriquecer las conclusiones extraídas. Los candidatos finales son *Naïve Bayes*, *Random Forest*, y *Deep Neural Network*.

3.2.1 *Naïve Bayes*

Los clasificadores bayesianos asignan la clase más probable a cada muestra, según la descripción que ofrece el vector de valores de sus variables. En su versión más simplificada, se

asume que las variables son independientes, es decir, se facilita la aplicación del Teorema de Bayes, descrito en las figuras 3.4 y 3.5 [22]. A pesar de esta asunción poco realista, la familia de clasificadores que surge de la premisa anterior, conocida como NB (*Naïve Bayes*), ha obtenido resultados positivos, compitiendo a menudo con alternativas más sofisticadas [23]. De hecho, y en contra de lo que podríamos intuir, hay estudios que indican que los clasificadores bayesianos funcionan bien, en según qué casos, incluso con dependencias fuertes en su conjunto de atributos [24].

$$\mathbf{p(yi \mid x1, x2, \ldots, xn)} = \frac{\mathbf{p(x1, x2, \ldots, xn \mid yi).p(yi)}}{\mathbf{p(x1, x2, \ldots, xn)}}$$

Figura 3.4: Probabilidad de una clase dados unos valores de sus variables según el Teorema de Bayes. p(x1) representa la probabilidad de que la variable x1 tome un valor; p(yi) representa la probabilidad de una clase; p(yi | x1,x2,...,xn) representa la probabilidad de una clase dado un conjunto de valores de las variables; p(x1, x2,..., xn | yi) representa la probabilidad de una combinación específica de valores de las variables dada una clase.

$$\mathbf{p(x1, x2, \ldots, xn \mid yi)} = \boldsymbol{p(x1 \mid yi)} \, . \boldsymbol{p(x2 \mid yi)} . \ldots . \boldsymbol{p(xn \mid yi)}$$

Figura 3.5: Versión simplificada del Teorema de Bayes bajo la asunción de que las variables son independientes.

Existen tres variantes en la familia de clasificadores NB. Se diferencian en que cada una de ellas espera una distribución determinada de sus variables:

- *Multinomial Naïve Bayes*: espera que sus variables sigan una distribución multinomial.
- *Bernoulli Naive Bayes*: espera que sus variables sigan una distribución discreta de Bernoulli.
- *Gaussian Naïve Bayes*: espera que sus variables sigan una distribución normal.

Por último, hablaremos de ventajas y desventajas de los clasificadores NB. Por un lado, algunas de las ventajas son:

- Tanto el entrenamiento como la predicción es rápida.
- Proceso de entrenamiento sencillo de comprender.
- Trabajan bien con conjuntos de datos de alta dimensionalidad.
- Robustos a la modificación de sus parámetros.

Por el otro, entre los inconvenientes de usar NB contamos con:

- Por definición, no puede aprovechar las interacciones entre variables.
- Es sensible a conjuntos sesgados, es decir, cuando el conjunto de entrenamiento no representa la distribución de clases en el conjunto global.

3.2.2 *Random Forest*

Los árboles de decisión son usados ampliamente tanto para tareas de clasificación como de regresión. Su funcionamiento se basa, a grosso modo, en aprender una jerarquía de preguntas condicionales, que los lleven a una decisión sobre la variable objetivo [25]. La desventaja principal de estos, es que tienden a ajustarse demasiado al conjunto de entrenamiento, perdiendo capacidad de generalización. RF (*Random Forest*) es la solución a este problema.

RF consiste en crear una colección de árboles de decisión, en la que cada árbol es ligeramente diferente a los demás. El razonamiento detrás de esto es que aunque cada árbol es capaz de hacer un buen trabajo prediciendo, también se ajusta demasiado a cierta parte de los datos. No obstante, si construimos suficientes árboles, se sobreajustarán en diferentes partes de los datos, de manera que podremos reducir la cantidad de sobreajuste del conjunto, haciendo un promedio de sus resultados. A esta técnica se la conoce como *bagging* (Bootstrap Aggregation). Esta reducción en el sobreajuste, a la vez que mantenemos el poder predictivo de los árboles, es demostrable matemáticamente. [25].

Para implementar esta estrategia, debemos crear un gran número de árboles de decisión. Además, los árboles han de ser diferentes entre sí. Para garantizar la diferencia, inyectamos aleatoriedad en la construcción de cada árbol, para asegurarnos de que será único. Hay dos maneras en las que podemos hacer esto: mediante la selección de las observaciones que emplearemos para construir los árboles, y mediante la selección de características disponibles para cada división de los mismos [25].

A la técnica de muestreo de las observaciones que empleamos para construir cada árbol se le denomina *bootstrapping*. Consiste en la selección aleatoria de muestras, con sustitución, del conjunto de entrenamiento. Como cada observación puede ser seleccionada más de una vez, habrá un porcentaje de ellas con las que cada árbol no habrá entrenado. Al conjunto de estas muestras se le conoce como OOB (*Out of Bag*), y suponen en torno a un 36.8% de todo el conjunto de entrenamiento. El conjunto OOB aporta un mecanismo de validación que podemos emplear si, por ejemplo, extraer una parte del conjunto de entrenamiento para crear un conjunto de validación no fuera viable. Sin embargo, cada muestra del OOB sólo puede ser usada para validar los árboles que no la hayan usado para entrenar, lo que reduce la potencia del *bagging*, y hace que la validación con OOB sea peor que el conjunto de validación. En la figura 3.6 se puede ver un diagrama de este proceso.

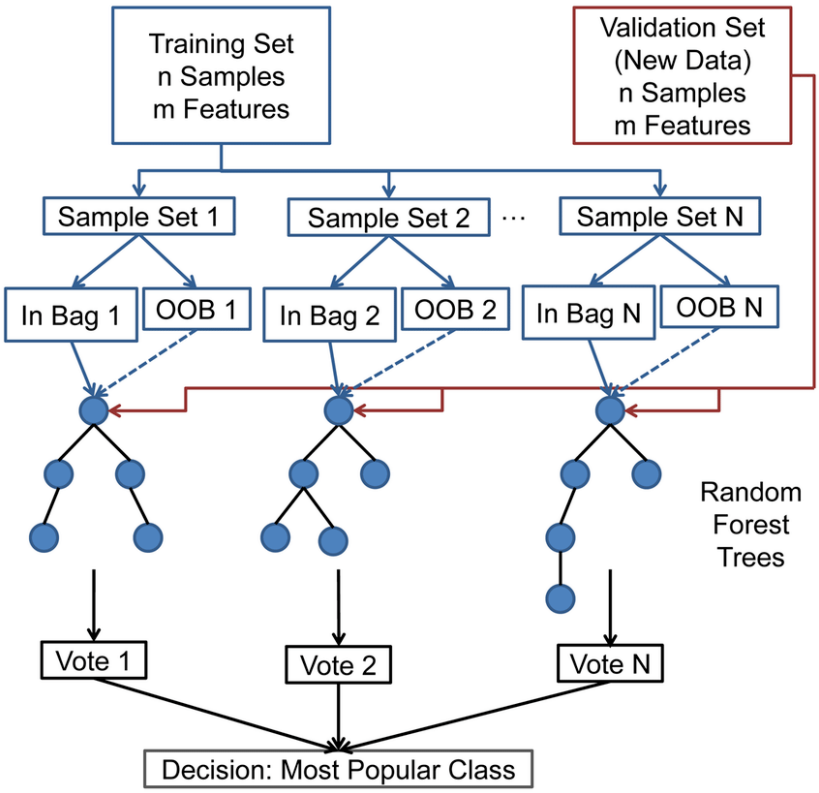


Figura 3.6: Funcionamiento de Random Forest.

Para terminar, en cuanto a los puntos fuertes de Random Forest tenemos que:

- Está entre los algoritmos más empleados tanto para regresión como para clasificación.
- No requiere mucha hiperparametrización.
- No requiere escalado de los datos.

Por otro lado, respecto a sus puntos débiles:

- Imposible de interpretar cuando el número de árboles es elevado.
- No suelen trabajar bien con conjuntos de datos con una alta dimensionalidad.

3.2.3 *Deep Neural Networks*

Las redes neuronales profundas, o deep neural networks (DNNs), son un tipo de arquitectura de DL. Son capaces de representar funciones de complejidad variable mediante la modificación del número de capas, y neuronas por capa, que la conforman. Se considera que una red neuronal es profunda si contiene más de tres capas, incluyendo la de entrada y la de

salida. Por lo tanto, cualquier red con al menos dos capas ocultas es considerada una DNN [20].

Se puede ver un ejemplo de la representación estándar de una arquitectura DNN en la figura 3.7. En esta, se esboza una red con varias capas totalmente conectadas. La notación que aparece en la figura es habitual en la representación de DNN [20].

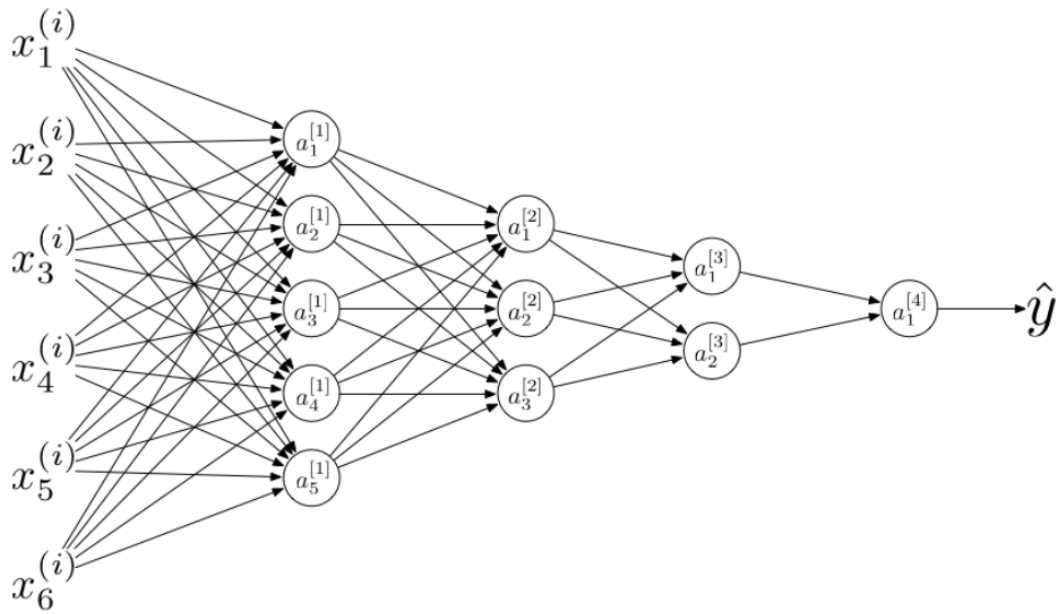


Figura 3.7: Ejemplo de arquitectura compleja de una DNN.

Uno de los factores que han propiciado el auge de este tipo de redes neuronales en los últimos años, son los avances e innovación en los algoritmos. Esto ha ayudado a hacer más eficiente y rápida la computación necesaria para las técnicas de DNN [20].

En cuanto a funciones de activación, en el pasado la sigmoidea fue la más empleada. Una desventaja de esta es que hay regiones de la función donde la pendiente es cercana a cero. Esto a menudo resulta en que los algoritmos tarden mucho en converger (concretamente en minimizar la pérdida). En consecuencia a lo anterior, la función ReLU ha ido volviéndose más popular [20].

Por último, como se ha hecho con los anteriores algoritmos, se destacarán algunas ventajas de trabajar con DNNs:

- Guardan información en toda la red en vez de en un solo punto.
- Son tolerantes a fallos.
- Excepcional capacidad de generalización.

- Se puede paralelizar el entrenamiento.

Respecto a los puntos negativos:

- Es imposible entender por qué funciona la red.
- Son dependientes del hardware.
- Requieren representaciones concretas de las variables para poder aprender de los datos.

3.3 *Dataset*

La baja tasa de implantación de los IDS se asocia, entre otras razones, a la complejidad de estos sistemas. Antes de poder ser desplegados, requieren pasar por numerosas pruebas, configuración, y evaluación [26].

En el pasado hubo dificultades produciendo conjuntos de datos públicos y adecuados para llevar a cabo la evaluación. Pese a que se hicieron algunas aportaciones significativas como los conjuntos de DARPA o KDD, su precisión y habilidad de reflejar la realidad fue criticada. La necesidad de conjuntos nuevos y más dinámicos, que reflejaran las tendencias de los patrones de amenazas y tráfico según fueran evolucionando, propiciaron la aparición de un nuevo paradigma [26].

Se trata de un acercamiento sistemático para crear conjuntos que sirvan para analizar, probar, y evaluar los sistemas de detección de intrusiones, centrándose en detección de anomalías en red. Como base se emplean los perfiles, que contienen una representación abstracta de eventos y comportamientos detectados. Estos cuentan con diversas implementaciones y son combinables, permitiendo a los investigadores crear conjuntos diversos. Según este paradigma se creó el UNB ISCXIDS2012, el conjunto que emplearemos en este proyecto [26].

3.3.1 UNB ISCXIDS2012

El proceso pormenorizado de la creación de este conjunto comenzó con la definición de perfiles para encapsular la manera en la que los usuarios se comportaban, y poder recrearla. A continuación, se programaron agentes para ejecutarlos, imitando la actividad de usuario. Después se diseñaron escenarios de ataque, donde se reflejaban casos reales de comportamiento malicioso. Estos fueron perpetrados en tiempo real desde dispositivos físicos con ayuda humana. De esa manera se evitó tener que unir a posteriori los conjuntos de tráfico normal y ataques, lo que podría llegar a inyectar características indeseadas. El resultado tiene la ventaja obvia de que puede ser etiquetado, lo que simplifica la evaluación de los IDS, y provee unos puntos de referencia más realistas. Además, el conjunto se ofrece tanto como capturas de tráfico, como agrupado en flujos ya etiquetados, siendo el último formato el que emplearemos nosotros. Podemos ver la distribución de la red que se emula en la figura 3.8 [26].

Los ataques con representación en el UNB ISCXIDS2012 son: infiltración en la red desde dentro, denegación de servicio HTTP, denegación de servicio distribuida (DDoS) empleando una *IRC Botnet*, y ataques SSH usando fuerza bruta [26].

Por último, es relevante destacar que existen versiones más actuales del conjunto que hemos decidido emplear. La razón principal por la que no las hemos usado es que consideramos que para los objetivos de este trabajo es más relevante la extensa documentación y casos de uso que existen sobre el UNB ISCXIDS2012.

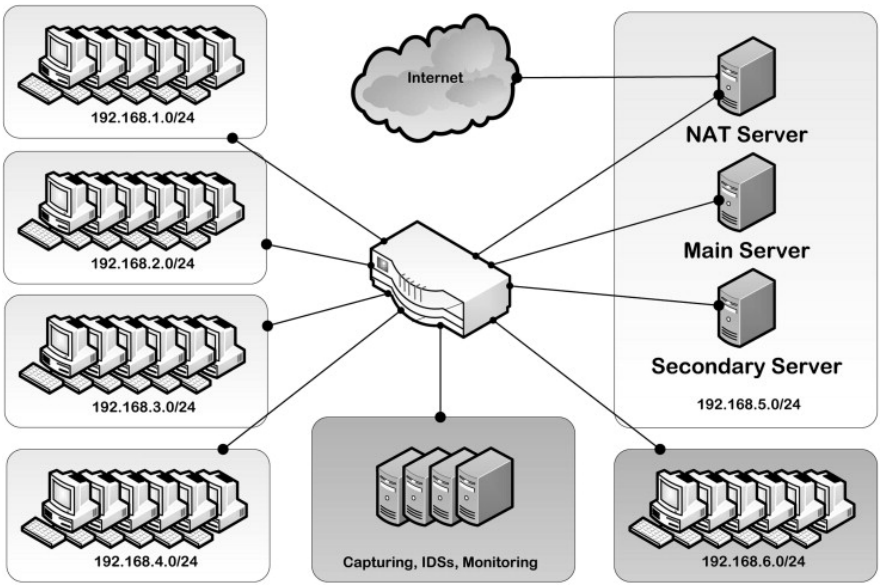


Figura 3.8: Diagrama de la arquitectura de red en la que se generó el conjunto de datos UNB ISCXIDS201.

3.4 IDS y NTA

Las metodologías de detección de intrusiones se pueden clasificar en cuatro grandes categorías [27]:

- Detección basada en firmas o *signature-based detection* (SD): una firma es un patrón que se corresponde con una amenaza conocida. SD es el proceso de comparar estos patrones contra los que existan en los eventos de nuestra red, de manera que se puedan reconocer posibles intrusiones. Como emplea conocimiento acumulado sobre ataques y vulnerabilidades específicas, a la SD también se la conoce como *knowledge-based detection* o *misuse detection*.

- Detección basada en anomalías o *anomaly-based detection* (AD): este tipo de metodología utiliza perfiles de tráfico normal para reconocer ataques. También se la conoce como

behaviour-based detection.

- Análisis de protocolos con estado o *stateful protocol analysis* (SPA): el IDS conoce y rastrea los estados de los protocolos. El proceso de SPA se parece al de AD, siendo la diferencia principal que AD emplea perfiles específicos de máquinas o de redes precargadas, mientras que SPA depende de los perfiles generales de los protocolos de nuestra red. SPA también es conocido como *specification-based detection*.

- Detección híbrida o *hybrid-most detection* (HD): se emplean varias metodologías para obtener una detección mayor y más precisa. Por ejemplo, SD y AD son complementarias, ya que la primera se centra en amenazas concretas, mientas que la última lo hace en ataques desconocidos.

Además, el estudio de la detección de intrusiones es tradicionalmente abordado desde dos perspectivas: detección de anomalías o *anomaly detection*, y *misuse detection*. Sin embargo, algunos autores consideran que no están correctamente diferenciadas. Existe una clasificación alternativa que propone cinco subclases bien definidas [27]:

- Basada en estadísticas o *statistics-based*: se hace uso de umbrales predefinidos, medias, desviaciones típica y probabilidades para identificar las intrusiones.

- Basada en patrones o *pattern-based*: se buscan ataques cuya firma se conoce.

- Basada en el estado o *state-based*: emplea una máquina de estados finita creada a partir del contexto de la red para identificar ataques.

- Basada en heurísticas o *heuristic-based*: está inspirada en conceptos de biología e inteligencia artificial.

Por otro lado, hoy en día existen muchos tipos de tecnología de IDS. Una categorización clásica las separa en cuatro clases según dónde se despliegan, y qué tipo de eventos de red reconocen [27]:

- IDS basado en host o *host-based IDS* (HIDS): monitoriza y recopila las características de las máquinas que contienen información importante, servidores encargados de servicios públicos, etc.

- IDS basado en red o *network based IDS* (NIDS): captura el tráfico de red en segmentos específicos a través de sensores, y a continuación analiza las actividades de las aplicaciones y protocolos para reconocer incidentes sospechosos.

- IDS basado en red inalámbrica o *Wireless-based IDS* (WIDS): es similar al NIDS, pero captura tráfico de redes inalámbricas.

- IDS híbrido o *mixed IDS* (MIDS): mediante la adopción de varias tecnologías simultáneamente, puede conseguir una detección más precisa y completa.

- IDS basado en el análisis del comportamiento de red o *network behaviour analysis* (NBA): inspecciona el tráfico de red para reconocer ataques a partir de flujos sospechosos.

En los últimos años, las nuevas necesidades que han ido surgiendo en nuestras redes para garantizar su seguridad, han afectado a la anterior clasificación de tecnologías. Así, partiendo de NBA, ha surgido una nueva tecnología llamada NTA (*Network Traffic Analysis*) [28]. NTA se caracteriza por tener una incidencia más transversal en las perspectivas que emplea, ya que usa un acercamiento híbrido que le permite adaptarse mejor a la evolución del perímetro de red. Según el *Market Guide for Network Traffic Analysis* de Gartner, publicado en febrero del 2019 [29], lo que diferencia a NTA es que emplea una combinación de ML, analítica avanzada, y detección basada en reglas, para detectar actividades sospechosas en las redes corporativas. Las herramientas NTA analizan continuamente paquetes o flujos para construir modelos que reflejen el comportamiento normal de la red. Además de monitorizar el tráfico vertical que cruza el perímetro empresarial, las soluciones NTA también pueden monitorizar las comunicaciones horizontales mediante el análisis del tráfico de red, agrupado en flujos o no, que recibe de sensores estratégicamente dispuestos.

3.5 Python/PySpark

Python lleva años siendo ampliamente utilizado en diversas áreas. Su comunidad activa, facilidad de uso, flexibilidad, y extensa cantidad de librerías, son algunas de las razones principales. Concretamente, en el ámbito de la ciencia de datos, estudios recientes han establecido que es el lenguaje de programación preferido [30].

Además, PySpark es uno de los lenguajes de programación con los que se puede trabajar en el *framework* Apache Spark, que usamos para la parte de distribución de este proyecto. PySpark es la API de Python que soporta Apache Spark, y nos permitirá programar de una manera muy similar a como lo hacemos con Python, pero haciendo un código que se pueda distribuir.

3.6 Scikit-learn

Scikit-learn es un módulo de Python que integra un amplio rango de algoritmos de ML, para problemas de aprendizaje supervisado y no supervisado. Lo hace a través de su interfaz, empleando un lenguaje de alto nivel y de propósito general. Algunas de las características de

este módulo incluyen: facilidad de uso, calidad del código, colaboración, buena documentación, y rendimiento. Además, apenas tiene dependencias y se distribuye a través de una licencia BSD simplificada, lo que propicia su uso tanto en el ámbito académico como comercial. Como se basa en el ecosistema Python, puede ser fácilmente integrado con otras aplicaciones fuera del entorno de la ciencia de datos.

3.7 Tensorflow y Keras

Scikit-learn no soporta técnicas de DL o *reinforcement learning*, ya que tienen requisitos especiales que no se alinean con sus restricciones de diseño. Para poder usar esas técnicas, hemos tenido que buscar una alternativa, en este caso TensorFlow y Keras.

TensorFlow es una plataforma *open-source* de ML. Podemos pensar en esta como en una capa de infraestructura. Keras es una API de alto nivel para TensorFlow, con una interfaz amigable y altamente productiva para resolver problemas de ML mediante el empleo de técnicas de DL. Fue desarrollada en torno a la base de pasar lo antes posible de tener una idea a experimentar con ella, así que combina toda la potencia y funcionalidad que permite TensorFlow con la facilidad de uso.

3.8 Planificación del proyecto

3.8.1 Metodología

Es necesario establecer algún tipo de planificación que ponga en un marco temporal la realización de las fases propuestas por CRISP-DM. Sin embargo, en este trabajo hemos decidido no comprometernos con una metodología en concreto. La razón principal es que incluso para las opciones enfocadas a proyectos pequeños y de características similares, era necesario distorsionar las metodologías en gran medida para prescindir de procesos o artefactos innecesarios. Por eso, preferimos simplemente inspirarnos en las metodologías ágiles de desarrollo iterativo, y coger de estas las ideas más convenientes.

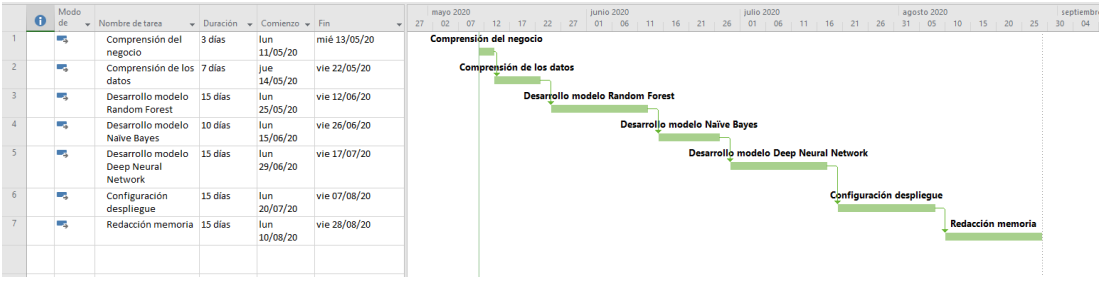
El equipo de trabajo lo conformaron Diego Fernández Iglesias, Francisco Javier Nóvoa Manuel, y Julio Jairo Estévez Pereira. Tanto Diego como Francisco tuvieron un rol de supervisores del desarrollo, estableciendo sus pautas y revisando el avance del mismo. El desarrollo lo llevó a cabo Julio.

La manera de proceder fue la siguiente: al comienzo del proyecto, tuvieron lugar unas primeras reuniones en las que el equipo de trabajo estableció los objetivos y requisitos generales que se debían alcanzar. A partir de entonces, se realizaron reuniones semanales en las que se revisaron los avances, y se plantearon las siguientes tareas concretas a abordar.

3.8.2 Planificación

Las tareas en las que se ha dividido el proyecto son: *comprensión del negocio*, *comprensión de los datos*, *desarrollo modelo Random Forest*, *desarrollo modelo Naïve Bayes*, *desarrollo modelo Deep Neural Network*, *configuración despliegue*, y *redacción memoria*.

La elección de estas actividades pretende ser lo más permisiva posible con la naturaleza iterativa de CRISP-DM. Así, en vez de tener una tarea por etapa del ciclo de vida CRISP-DM, establecemos una tarea de desarrollo de cada una de las técnicas de ML que se ha decidido emplear, y damos por hecho que, para cada tarea, tendrá lugar una iteración de todas las fases intermedias. Es decir, la actividad *desarrollo modelo Random Forest* contiene su correspondiente preparación de los datos, modelado y evaluación. En la figura 3.9, se pueden ver las actividades dispuestas en un diagrama de Gantt.



	Modo de	Nombre de tarea	Duración	Comienzo	Fin
1		Comprensión del negocio	3 días	lun 11/05/20	mié 13/05/20
2		Comprensión de los datos	7 días	jue 14/05/20	vie 22/05/20
3		Desarrollo modelo Random Forest	15 días	lun 25/05/20	vie 12/06/20
4		Desarrollo modelo Naïve Bayes	10 días	lun 15/06/20	vie 26/06/20
5		Desarrollo modelo Deep Neural Network	15 días	lun 29/06/20	vie 17/07/20
6		Configuración despliegue	15 días	lun 20/07/20	vie 07/08/20
7		Redacción memoria	15 días	lun 10/08/20	vie 28/08/20

Figura 3.9: Diagrama de Gantt de las tareas del proyecto.

3.8.3 Seguimiento

En el diagrama de la figura 3.10, se puede observar el seguimiento del proyecto. La primera actividad, *comprensión del negocio*, se llevó a cabo en tiempo. Respecto a la *comprensión de los datos*, pudo recortarse parte del tiempo planificado porque el conjunto de datos no contaba con un número muy grande de atributos. Con la tarea *desarrollo modelo Random Forest* hubo un retraso considerable debido a que se encontraron resultados no esperados que se tuvieron que contrastar. Por otro lado, la actividad *desarrollo modelo Naïve Bayes* resultó ser más rápida de acometer de lo esperado. Para la tarea *desarrollo modelo Deep Neural Network* se pudieron acortar unos días, aunque su dificultad se asemejaba a la estimación. La actividad *configuración despliegue* supuso algunos retos, documentados más adelante, que requirieron dedicarle tiempo extra. Finalmente, también resultó ser ligeramente optimista la estimación para la actividad *redacción memoria*.

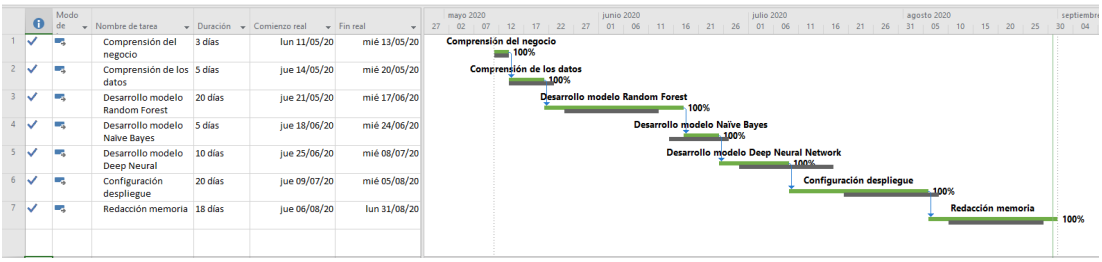


Figura 3.10: Vista del seguimiento del proyecto.

3.8.4 Estimación de costes

Para estimar los costes del proyecto se ha dado por hecho que el equipo está formado por un investigador y un jefe de proyecto. En ese caso, y teniendo en cuenta los salarios medios en España de cada uno [31], obtenemos un coste de 14€/h para el investigador y 14,89€/h para el jefe de proyecto. Estableciendo una jornada diaria de 4,5 horas para el investigador, el cálculo de coste para cada tarea de la planificación pueden observarse en la tabla 3.1. Además, estimando que el jefe de proyecto ha trabajado 60 horas en total en el proyecto, podemos estimar los costes totales en la tabla 3.2.

Tarea	**Esfuerzo (h)**	**Coste (€)**
Comprensión del negocio	13,5	189
Comprensión de los datos	22,5	315
Desarrollo modelo Random Forest	90	1.260
Desarrollo modelo Naïve Bayes	22,5	315
Desarrollo modelo Deep Neural Network	45	630
Configuración despliegue	90	1.260
Redacción memoria	81	1.134

Tabla 3.1: Coste de cada tarea según la estimación salarial del investigador.

Puesto	**Coste (€)**
Investigador	5.103
Jefe de proyecto	893,4
TOTAL: 5.996,4 €	

Tabla 3.2: Coste total del proyecto según las estimaciones salariales de los dos perfiles.

Capítulo 4

Comprensión de los datos

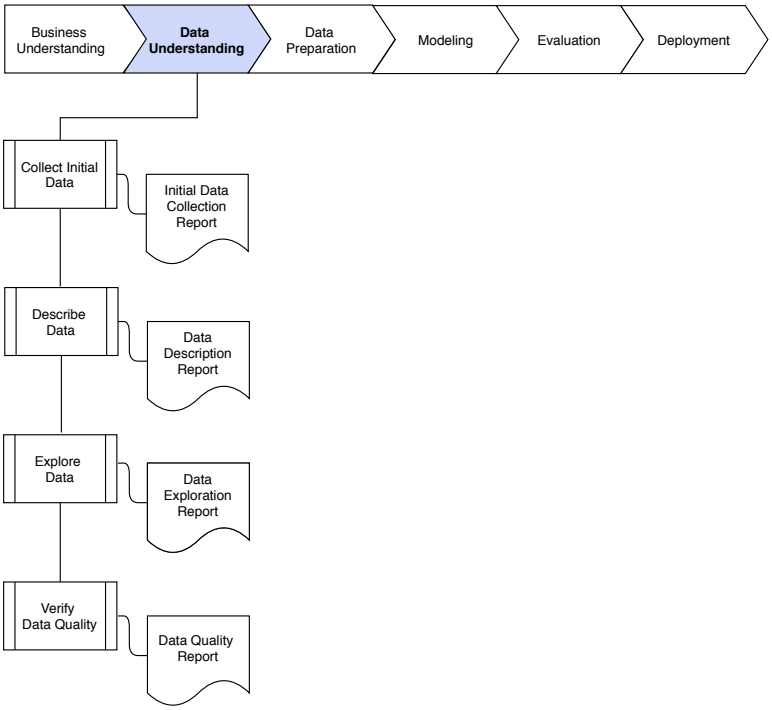


Figura 4.1: Tareas de la fase dos de CRISP-DM: comprensión de los datos.

En este capítulo hablaremos de la segunda etapa del ciclo de vida CRISP-DM. Como se puede ver en la figura 4.1, las actividades que la componen son:

- Recolección inicial de los datos: esta tarea comprende la adquisición de los datos necesarios, y si hace falta, su carga en nuestra herramienta de análisis. Se aborda en la sección 4.1.
- Descripción de los datos: esta tarea comprende examinar superficialmente los datos y documentar sus características generales. Ya se realizó en la sección 3.3.

- Exploración de los datos: esta tarea comprende profundizar en nuestros datos y sus características. Concretamente incluye consultar, visualizar y buscar relaciones. Se aborda en la sección 4.2.
- Verificación de la calidad de los datos: esta tarea incluye comprobar si los datos son completos, correctos, y si faltan valores, para asegurarnos de que sólo usamos información que refleje la realidad. Se aborda en la sección 4.3.

4.1 Recolección

La creación de un conjunto de datos propio se sale del alcance de este trabajo, así que decidimos hacer uso de uno ya generado. De entre las opciones disponibles, elegimos el UNB ISCXIDS2012, descrito en la sección 3.3. Por lo tanto, en esta fase sólo cargamos el conjunto de datos desde los archivos.

4.2 Exploración

En esta sección nos enfocamos en conocer mejor nuestros atributos y sus relaciones subyacentes. Partimos del conjunto de datos recién cargado. El *dataset* original tiene 22 atributos, que se pueden observar en la figura 4.2a. También se puede ver los detalles de su distribución en el anexo A.2. Nosotros nos centraremos en los que en la sección 4.3 se catalogan como relevantes, listados en la figura 4.9. Lo primero que hacemos es clasificar las variables, en diferentes grupos, con respecto a la naturaleza de sus distribuciones. De esta manera, podemos decidir cuáles son las representaciones, para la visualización, más adecuadas para cada grupo en vez de individualmente. Hacemos 3 grupos:

- Grupo 1:
  - *'totalSourceBytes'*
  - *'totalDestinationBytes'*
  - *'totalDestinationPackets'*
  - *'sourcePort'*
  - *'destinationPort'*
- Grupo 2:
  - *'source'*
  - *'destination'*

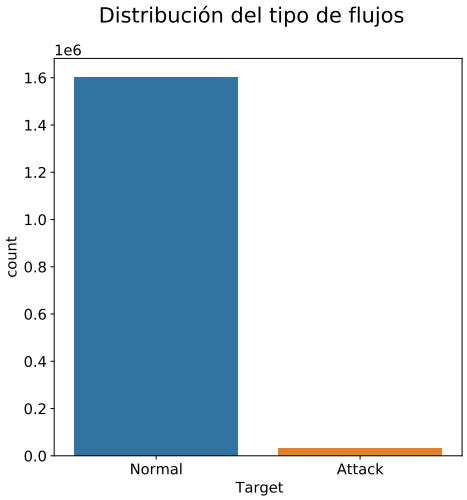
- Grupo 3:
  - *'sourceTCPFlagsDescription'*
  - *'destinationTCPFlagsDescription'*
  - *'protocolName'*
  - *'direction'*

Como se puede apreciar, los atributos 'startDateTime' y 'stopDateTime' no aparecen en ningún grupo, ya que aunque son interesantes para la ingeniería de características que se realizará en apartados posteriores, por sí solos no son más que una secuencia de marcas temporales.

Es relevante mencionar que los resultados de este apartado están condicionados por la distribución de tipos de flujo. Como podemos ver en la figura 4.2b, existe un gran desajuste entre el número de flujos de ataque y el número de flujos normales.

```
appName                          object
totalSourceBytes                  int64
totalDestinationBytes             int64
totalDestinationPackets           int64
totalSourcePackets                int64
sourcePayloadAsBase64            object
sourcePayloadAsUTF               object
destinationPayloadAsBase64       object
destinationPayloadAsUTF          object
direction                        object
sourceTCPFlagsDescription        object
destinationTCPFlagsDescription   object
source                           object
protocolName                     object
sourcePort                        int64
destination                      object
destinationPort                   int64
startDateTime                    object
stopDateTime                     object
sensorInterfaceId               float64
startTime                       float64
Target                           object
dtype: object
```

(a) Atributos originales del *dataset* completo.



(b) Gráfica del número de observaciones de flujos normales vs. número de observaciones de flujos de ataque.

Figura 4.2: Información general del *dataset* original. A la izquierda, sus atributos originales, a la derecha, la distribución de tipos de flujos.

4.2.1 Grupo 1

Los atributos en este grupo son continuos. Además, podemos clasificarlos según la información que contienen, como relativos a la cantidad de información enviada en cada flujo, y relativos a los puertos empleados en cada flujo. Esta diferenciación es relevante porque los

primeros pueden tomar valores positivos sin un límite superior concreto, mientras que los segundos sólo pueden ser enteros entre 0 y 65535 [32].

Para poder entender cuál es la relación de los atributos de este grupo con la variable objetivo, hacemos gráficas de distribución de valores respecto a la variable objetivo *'Target'*. Podemos verlas en la figura 4.3. A partir de estas gráficas concluimos que hay atributos cuya distribución parece estar bastante relacionada con si un flujo es parte de un ataque o no. El caso más claro es el de *'sourcePort'*, en el que según el rango de valores en el que nos encontremos, tenemos una predominancia clara de un tipo de flujo u otro.

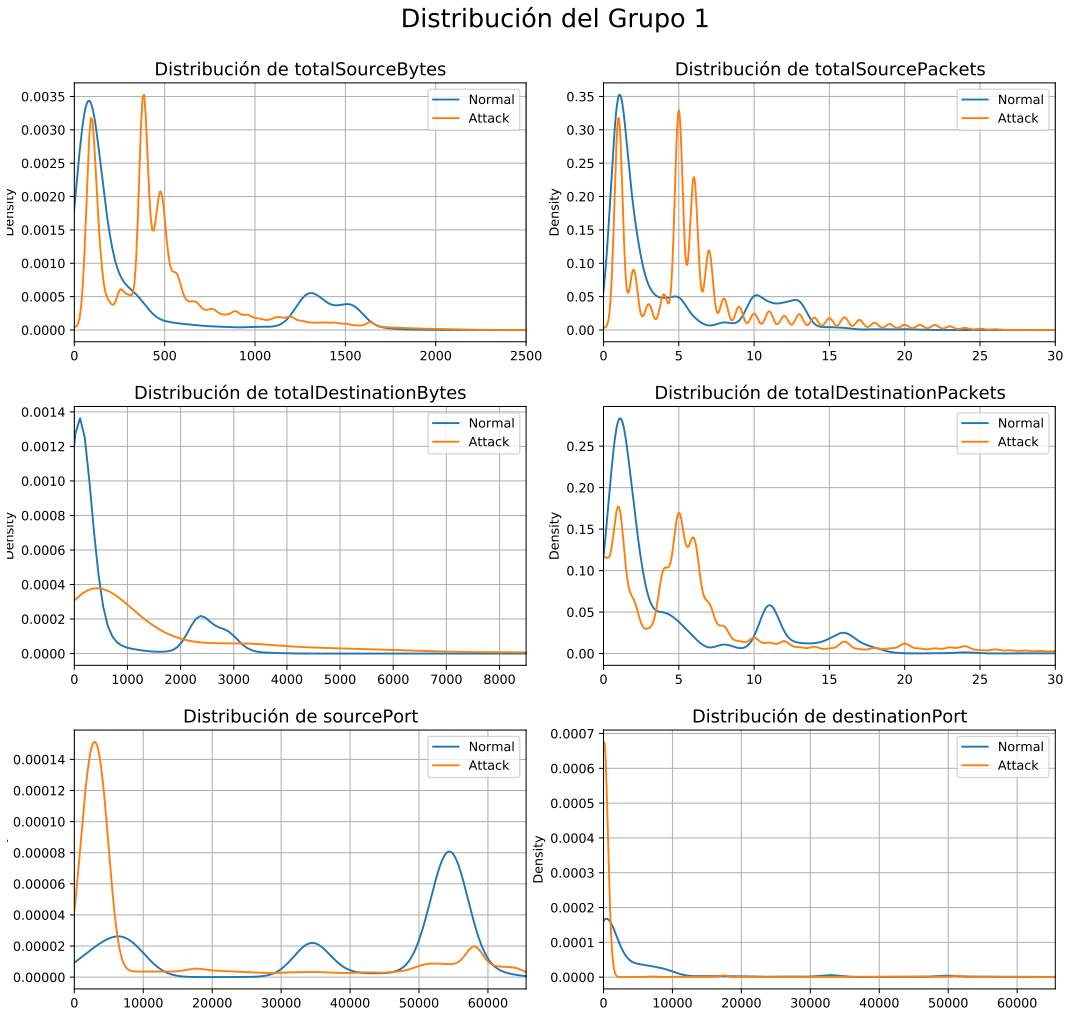


Figura 4.3: Gráficas de la distribución de los atributos del Grupo 1 con respecto a *'Target'*.

4.2.2 Grupo 2

En este grupo tenemos las variables *'source'* y *'destination'*, que contienen la IP origen y destino del flujo. Esta es una información de naturaleza categórica y de alta cardinalidad. Para facilitar la extracción de información y visualización, se decide agrupar las IPs de ambos atributos en base a su subred, resultando en las siguientes categorías: *'192.168.1'*, *'192.168.2'*, *'192.168.3'*, *'192.168.4'*, *'192.168.5'*, *'192.168.6'*, *'others'* y *'external'*. Esta agrupación se hace atendiendo a la arquitectura descrita en la figura 3.8, siendo la categoría *'external'* para las IPs de fuera de la red, y *'others'* para todas las demás no pertenecientes a las subredes definidas (IPs privadas, APIPA, etc.). Esto nos ayuda a trasladar el espacio de tan alta cardinalidad del que partimos, en uno con sólo 8 categorías, con lo que la visualización se vuelve abordable. En la figura 4.4, y figura 4.5, podemos ver cómo se distribuyen las IPs de *'source'* y *'destination'* con respecto al tipo del flujo.

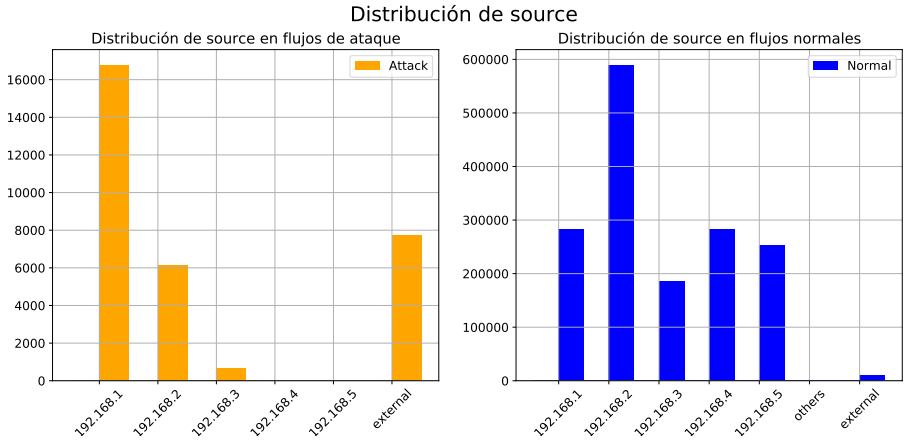


Figura 4.4: Gráficas de la distribución de *'source'* con respecto a *'Target'*.

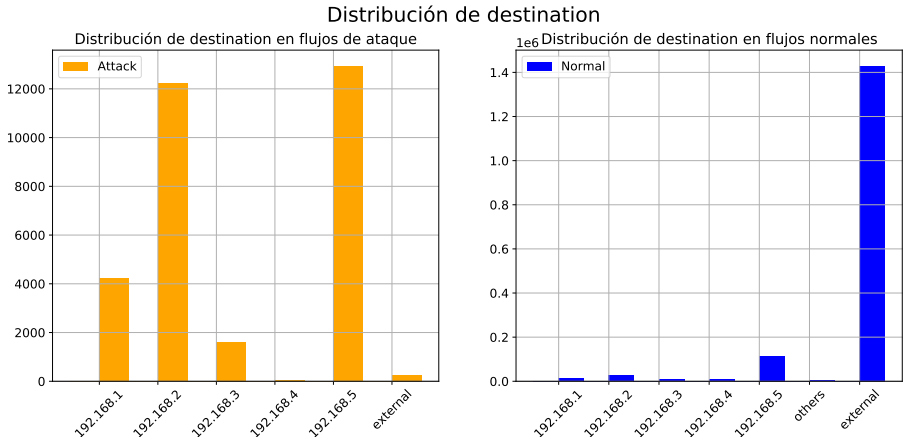


Figura 4.5: Gráficas de la distribución de *'destination'* con respecto a *'Target'*.

También en este caso se pueden observar relaciones de los atributos con la variable objetivo. Por ejemplo, en *'source'*, que la IP esté en las subredes *'192.168.4'* y *'192.168.5'* es un buen indicador de que un flujo es de tráfico normal. Por lo contrario, en *'destination'*, que una IP esté en las subredes *'192.168.2'* o *'192.168.5'*, indica con gran probabilidad que un flujo es de ataque.

4.2.3 Grupo 3

El último grupo está compuesto por variables categóricas de baja cardinalidad. Concretamente: *'sourceTCPFlagsDescription*, 'destinationTCPFlagsDescription', 'protocolName' y 'direction'. Para cada una de estas, consideramos que es relevante saber cómo se relacionan sus categorías con el tipo de flujo. Visualizar lo anterior es posible, pero son necesarias numerosas gráficas. Por consiguiente, aquí sólo incluimos un caso en el que se aprecie bien lo que pretendemos. En la figura 4.6 podemos ver que si en *'direction'* tenemos el valor 'L2R' (Local to Remote), en muchos casos estamos ante un flujo normal. Esta relación es bastante intuitiva, ya que si lo pensamos, lo que nos están diciendo los datos es que los ataques no suelen estar perpetrados mediante flujos que se generan dentro de la red y están dirigidos hacia fuera. Por otro lado, esa es la razón de que en el resultado se vea una tendencia tan clara pese a que, como hemos comentado, esté especialmente condicionado por la diferencia de cantidad entre flujos de ataque y flujos normales que hay en el *dataset*.

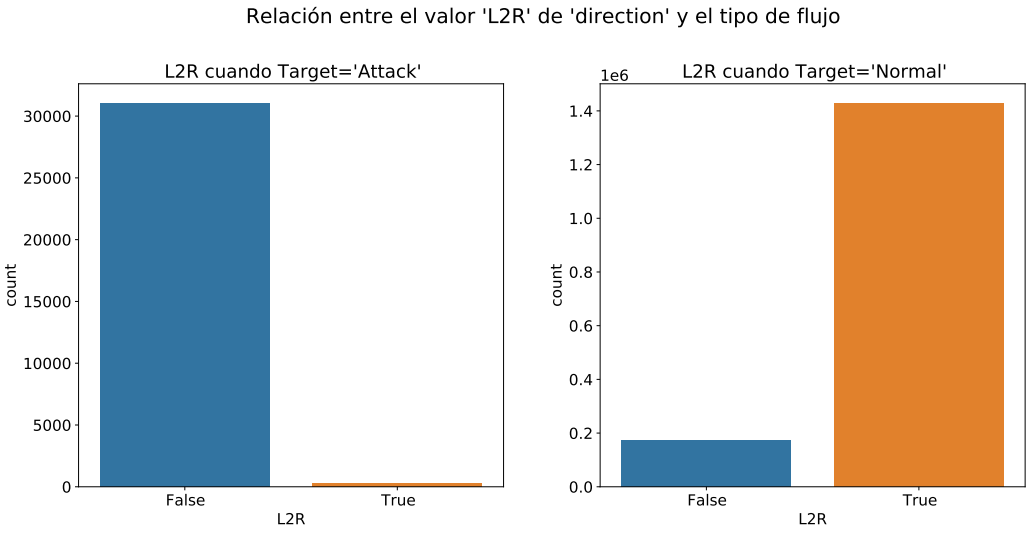


Figura 4.6: Gráficas de la distribución de *'L2R'* con respecto a *'Target'*.

4.2.4 Correlaciones

Después de analizar la relación de los atributos con respecto a la variable objetivo, calculamos la relación entre todos ellos. En la figura 4.7, y la figura 4.8, podemos ver dos mapas de correlación. Para hacer estos mapas, no empleamos todas las variables tal y como están en el *dataset* original, sino que realizamos algunas transformaciones similares a las empleadas en los dos apartados anteriores. Con ello pretendemos centrarnos más en describir las relaciones que hay entre los contenidos de las variables. Los cambios que hacemos, partiendo de los atributos de la figura 4.9 son:

- *'totalSourceBytes'*, *'totalSourcePackets*, *'totalDestinationBytes'*, *'totalDestinationPackets'*, *'sourcePort'*, *'destinationPort'* y *'Target'*: sin modificaciones.
- *'sourceTCPFlagsDescription'*, *'destinationTCPFlagsDescription'*, *'protocolName'* y *'direction'*: creamos una variable binaria por cada valor de estos atributos, que representan si ese valor aparece o no en la variable original.
- *'source'* y *'destination'*: agrupamos por subred, y volvemos a crear una variable binaria por cada valor.
- *'startDateTime'* y *'stopDateTime'*: no las tenemos en cuenta.

Estas modificaciones nos permiten saber qué categorías de nuestros atributos categóricos, ahora convertidas en atributos independientes, están relacionadas con otros atributos de nuestro conjunto de datos. Y en el caso de las IPs, los cambios nos dejan ver las relaciones a nivel de subred, en vez de que sirvan sólo como identificadores de máquinas. Las variables concretas originadas por las modificaciones son:

- De *'sourceTCPFlagsDescription'*: *'SA'*, *'SR'*, *'SS'*, *'SF'*, y *'SP*.
- De *'destinationTCPFlagsDescription'*: *'DA'*, *'DR'*, *'DS'*, *'DF'*, y *'DP*.
- De *'protocolName'*: *'tcp_ip'*, *'udp_ip'*, *'icmp_ip'* e *'igmp'*.
- De *'direction'*: *'R2L'*, *'L2R'* y *'L2L'*.
- De *'source'*: *'S192.168.1'*, *'S192.168.2'*, *'S192.168.3'*, *'S192.168.4'*, *'S192.168.5'*, *'S192.168.6'*, *'Sothers'* y *'Sexternal'*.
- De *'destination'*: *'D192.168.1'*, *'D192.168.2'*, *'D192.168.3'*, *'D192.168.4'*, *'D192.168.5'*, *'D192.168.6'*, *'Dothers'* y *'Dexternal'*.

Por último, los dos mapas están filtrados por correlaciones superiores a 0.6 e inferiores a -0.6 para ver sólo las más relevantes. Podemos comprobar que las correlaciones más fuertes son las esperadas. Por ejemplo, en las positivas, la nueva variable 'Sexternal', que indica si la IP origen de los flujos es de fuera de la organización, está totalmente correlacionada con la nueva variable 'R2L', que nos dice si el flujo se originó desde fuera hacia dentro de la red. Por el otro lado, en las correlaciones negativas, tenemos que 'Dexternal', que nos indica si el destino del flujo está fuera de la red, está inversamente correlacionada con 'L2L', que nos dice si el flujo tuvo origen y destino dentro de la red.

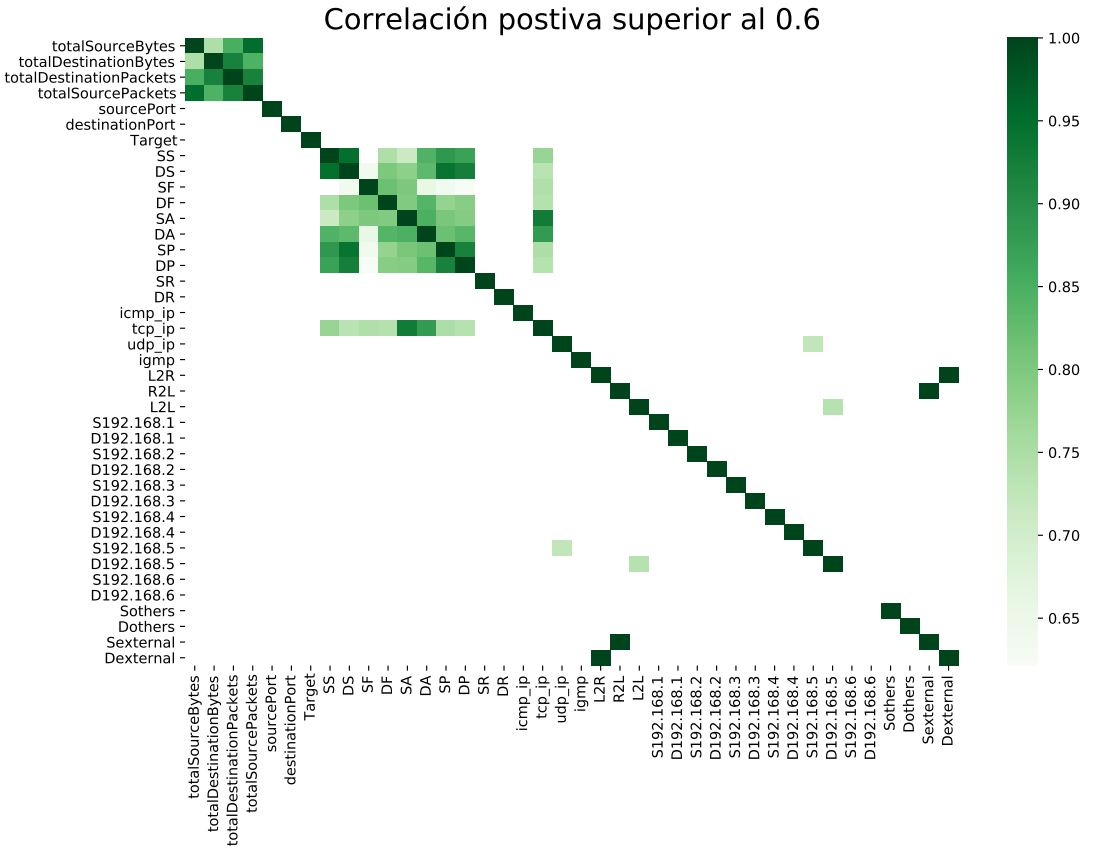


Figura 4.7: Mapa de correlaciones positivas mayores a 0.6 entre los atributos modificados de nuestro *dataset*.

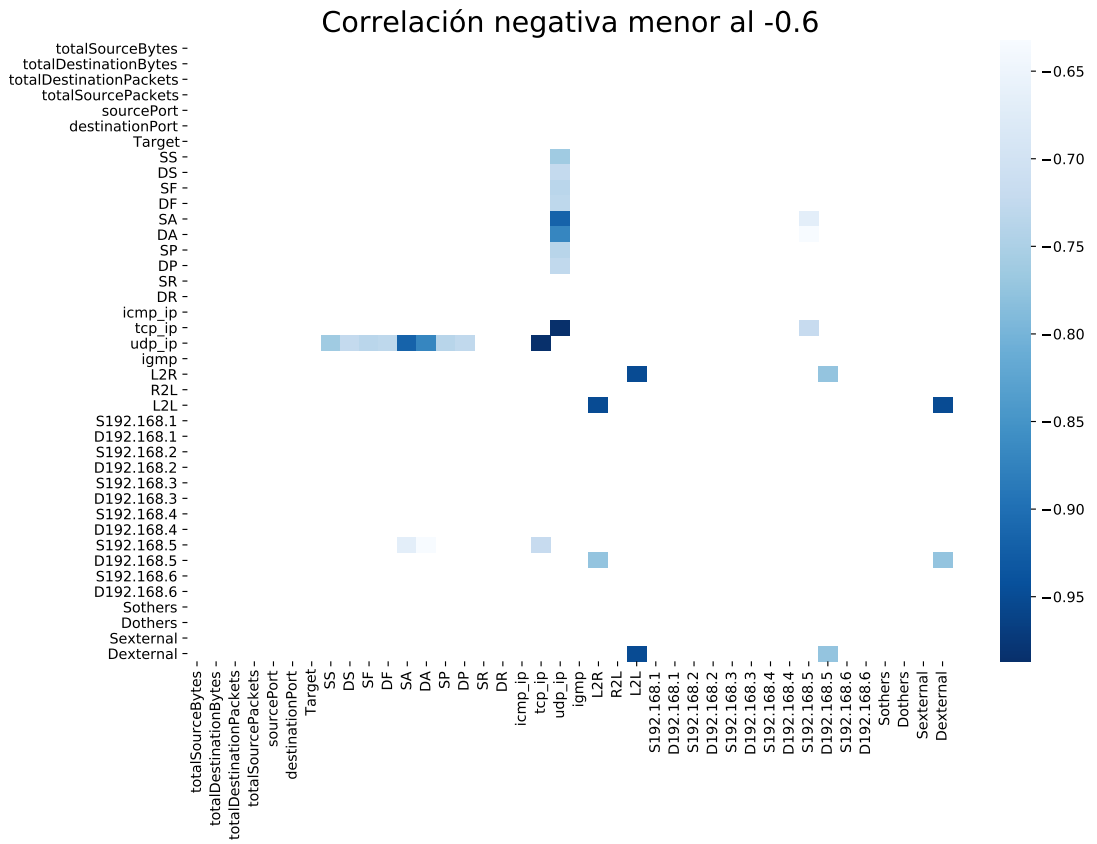


Figura 4.8: Mapa de correlaciones inversas menores a -0.6 entre los atributos modificados de nuestro *dataset*.

4.3 Verificación de la calidad

Los conjuntos de datos suelen contener valores erróneos y ausentes. En consecuencia, lo más habitual es que para poder usarlos eficazmente, haya que realizar una limpieza previa.

En este apartado haremos un informe de qué problemas hay en la calidad de los datos, y propondremos soluciones. Posteriormente, en la sección 5.1, estas proposiciones se tendrán en cuenta en la limpieza de los datos.

4.3.1 Datos erróneos

Atributos

Empezamos revisando la calidad de los atributos. Los dos primeros que evaluamos son *'sensorInterfaceId'* y *'startTime'*. Para explicar la razón, es relevante puntualizar que los archivos que contienen el conjunto de datos están divididos por días, y dependiendo del día, a

veces también en segmentos. Algunos de estos archivos contienen atributos espurios que no aparecen en ninguno de los demás. Esto significa que para las filas de todos los fragmentos del conjunto de datos menos uno, el valor de los atributos es nulo.

Respecto a la utilidad de los atributos sin tener en cuenta la elevada incidencia de nulos, *'sensorInterfaceId'* es potencialmente útil, ya que identifica la parte de la red donde el flujo ha sido capturado. Por el otro lado, *'startTime'* no aporta nada, principalmente porque entre los otros atributos ya contamos con *'startDateTime'*, que indica información similar y no tiene el problema de los nulos.

Valorando lo anterior, no cabe duda de que *'startTime'* debería ser eliminado. Y en este caso, también se recomienda la eliminación de *'sensorInterfaceId'*, porque consideramos que su utilidad no es suficiente como para compensar el número de valores ausentes.

Una vez examinados los atributos espurios de nuestro *dataset*, pasamos a buscar aquellos cuya información no nos es útil. En este grupo entran *'appName'*, *'sourcePayloadAsBase64'*, *'sourcePayloadAsUTF'*, *'destinationPayloadAsBase64'* y *'destinationPayloadAsUTF'*.

En cuanto a *'appName'*, nos indica información del nivel de aplicación en el modelo OSI, concretamente qué aplicación generó cada flujo. Sin embargo, en este trabajo lo que se analiza son los flujos, así que lo que nos interesa es la información de la capa de transporte e inferiores. Por lo tanto, proponemos su eliminación.

La situación es similar para *'sourcePayloadAsBase64'*, *'sourcePayloadAsUTF'*, *'destinationPayloadAsBase64'* y *'destinationPayloadAsUTF'*. Estos atributos nos dan el *payload* o carga útil, en los formatos base64 y UTF-8, de los paquetes de los flujos. Como nuestro análisis se centra en los flujos y no en los paquetes individuales, carece de interés la carga útil de estos últimos. En consecuencia, proponemos también la eliminación de estos atributos.

Si se aplican todas las modificaciones recomendadas, obtendríamos la lista reducida de atributos que podemos observar en la figura 4.9.

```
totalSourceBytes                int64
totalDestinationBytes           int64
totalDestinationPackets         int64
totalSourcePackets              int64
direction                      object
sourceTCPFlagsDescription      object
destinationTCPFlagsDescription object
source                         object
protocolName                   object
sourcePort                      int64
destination                    object
destinationPort                 int64
startDateTime                  object
stopDateTime                   object
Target                         object
dtype: object
```

Figura 4.9: Atributos del *dataset* después de eliminar los espurios e inútiles.

Observaciones

Tras asegurarnos de cuáles son los atributos relevantes, tenemos que comprobar si sus valores son siempre válidos. En caso de que no lo sean, se propondrán medidas para evitar que las observaciones defectuosas afecten a los resultados.

Para empezar, hablaremos de la presencia de la IP *0.0.0.0* en los atributos *'source'* y *'destination'*. Estos dos atributos nos indican la IP origen y destino de cada flujo. Sin embargo, la dirección *0.0.0.0* es no enrutable, y está designada para representar a un *host* inválido, desconocido, o no aplicable [33]. Por eso no tiene sentido tenerla identificando *hosts* en nuestro conjunto de datos, así que sugerimos la eliminacion de las filas en las que aparezca.

Por otro lado están los atributos 'sourceTCPFlagsDescription' y 'destinationTCPFlagsDescription', que contienen los *flags* TCP de cada flujo. Entre sus valores podemos encontrar *'Illegal7'* e *'Illegal8'*, que según la especificación, no son *flags* permitidos [34]. En esta ocasión no proponemos eliminar cada observación al completo si aparece uno de estos valores, como hicimos anteriormente, sino que la recomendación consiste en retirarlos de la lista de *flags*. De esa forma, no perdemos la información que nos aporta el flujo, y el resto de sus *flags* válidos.

Otro de los valores que puede aparecer listado en los atributos anteriores es el *flag 'U'* (URG). Por lo poco usado que es [35], y por el hecho de que siempre indique la presencia de un ataque en nuestro *dataset*, se sugiere tratarlo como un dato inválido, y retirarlo también de la lista.

Por último, el atributo *'destination'*, que nos indica la dirección de cada flujo, toma cuatro valores en nuestro conjunto de datos: *'L2L'* (Local to Local), *'L2R'* (Local to Remote), *'R2L'* (Remote to Local), y *'R2R'* (Remote to Remote). El último de estos valores representa una comunicación que tiene como origen un punto fuera de la red local, y como destino también un punto fuera de la red local. Esta situación no tiene sentido, porque si la comunicación ocurriese completamente fuera de la red, no habría sido capturada por nuestros sensores. Por lo tanto, recomendamos eliminar las observaciones con esa dirección.

4.3.2 Datos ausentes

Como muestra la figura 4.10, los atributos de nuestro conjunto de datos con valores nulos son *'sourceTCPFlagsDescription'* y *'destinationTCPFlagsDescription'*. Hay diversas opiniones respecto a cuál es el porcentaje de valores ausentes a partir del cual se deberían descartar las variables [36]. Sin embargo, en última instancia la decisión está supeditada al criterio del científico de datos. En este caso recomendaos mantener ambos atributos. La razón principal es que consideramos que un valor ausente representa información válida, concretamente, que el flujo no tiene *flags*.

Your selected dataframe has 15 columns.
There are 2 columns that have missing values.

	Missing Values	% of Total Values
destinationTCPFlagsDescription	460984	24.5
sourceTCPFlagsDescription	402290	21.4

Figura 4.10: Atributos del *dataset* con valores nulos, número de muestras y porcentaje.

Capítulo 5

Preparación de los datos

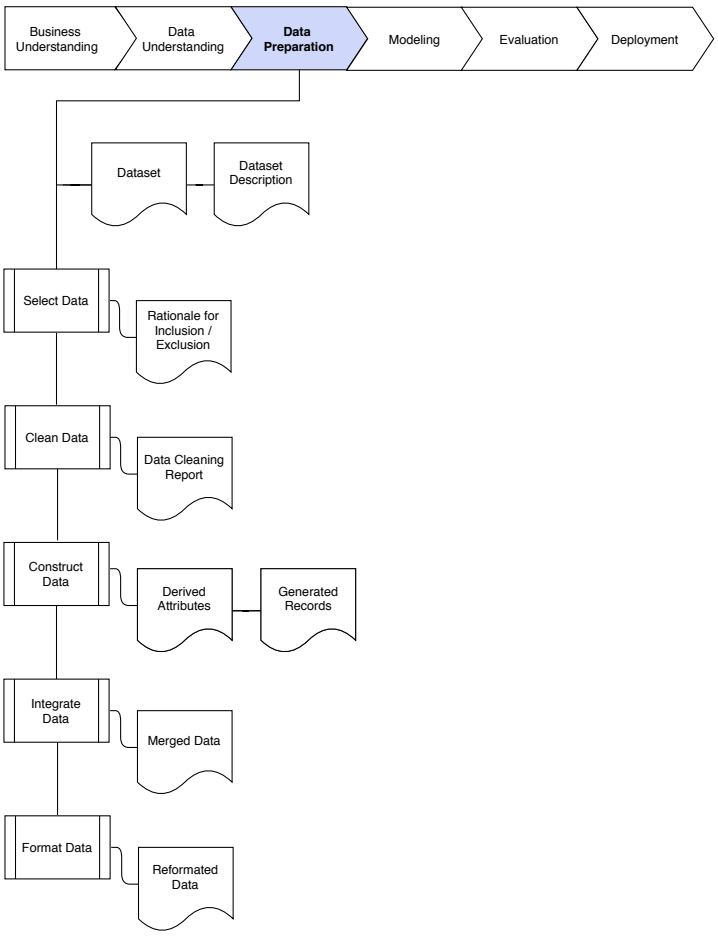


Figura 5.1: Tareas de la fase tres de CRISP-DM: preparación de los datos.

Este capítulo trata la tercera fase del ciclo de vida CRISP-DM. En la sección 4.3 elaboramos un informe de calidad, en el que se habla de la relevancia de los atributos, de los

problemas generales de calidad del *dataset*, y se proponen soluciones respecto a estos últimos. En esta fase utilizamos como conjunto de datos el resultado de aplicar al conjunto original las soluciones recomendadas. Como se puede observar en la figura 5.1, las actividades a realizar son:

- Selección de los datos: en esta tarea se elige la información que vamos a usar con cada técnica de ML. Se aborda en la sección 5.3.
- Limpieza de los datos: esta tarea consiste en asegurarnos de que los datos tienen, además de un nivel de calidad general, ya adquirido después de aplicar las modificaciones del informe de la sección 4.3, la calidad requerida por las técnicas de ML que vamos a usar. Se aborda en la sección 5.1.
- Construcción de datos: en esta tarea se realizan operaciones como la creación de nuevos atributos, o la transformación de atributos existentes. Se aborda en la sección 5.2.
- Integración de datos: esta tarea se basa en la aplicación de métodos que combinen información de varias fuentes, para la creación de nuevos atributos. Se aborda conjuntamente con la tarea de construcción en la sección 5.2.
- Formateado de datos: esta tarea incluye la modificación de la representación de los datos, sin cambiar su significado, para permitir o facilitar su uso en las técnicas que empleemos. Se aborda conjuntamente con las tareas de construcción e integración en la sección 5.2.

5.1 Limpieza

Muchos algoritmos de ML son sensibles al rango de valores de los atributos en el conjunto de datos.

Un punto atípico es una observación que está a una distancia anormal del resto. La presencia de este tipo de observaciones en los datos puede sesgar y confundir al proceso de entrenamiento, provocando tiempos de convergencia mayores, menos precisión, y en general, modelos que generen peores resultados.

En nuestro caso sólo consideramos relevante el análisis de valores atípicos en algunas de las variables numéricas. Concretamente en: *'totalSourceBytes'*, *'totalDestinationBytes'*, *'totalDestinationPackets'*, *'totalSourcePackets'*. No incluimos a los atributos *'sourcePort'* y *'destinationPort'*, porque ya sabemos que su rango de valores tiene que ir de 0 a 65535 [32].

Para llevar a cabo la limpieza de anomalías en los datos emplearemos *Isolation Forest*. Sus ventajas principales son que no se necesita escalar los datos de los atributos que se le

pasan; que tiene pocos parámetros, lo que lo hace robusto y fácil de optimizar; y que Scikit-learn cuenta con una implementación sencilla y bien documentada. Además, consideramos que es posible que haga un mejor trabajo encontrando los valores atípicos que un método más sencillo como el rango intercuartílico, que se basa únicamente en eliminar los valores por encima y debajo de un valor umbral.

5.1.1 *Isolation Forest*

Isolation Forest, a partir de ahora IF, es una técnica de ML basada en conjuntos de árboles de decisión, que se emplea para la detección de puntos anómalos en los datos. Lo que diferencia a IF de otros métodos de detección de valores atípicos, es que identifica las anomalías explícitamente en vez de hacerlo perfilando las observaciones regulares [37].

IF parte de la idea de que las anomalías en nuestros datos son menos frecuentes que las observaciones normales, y además están alejadas del *cluster* que conforman estas últimas. Por eso, empleando árboles con un particionado aleatorio, deberían encontrarse en menos divisiones, o lo que es lo mismo, aislar un valor atípico requiere menos condiciones que una observación normal [37]. Podemos ver una ilustración de esta idea en la figura 5.2 [38].

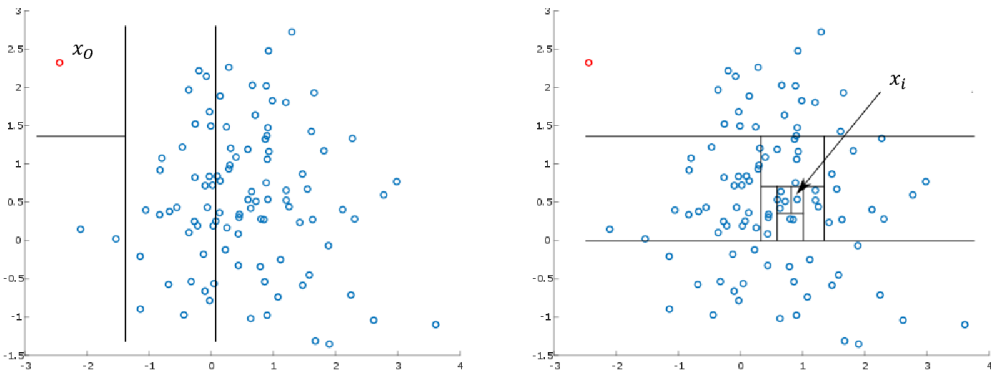


Figura 5.2: Particiones aleatorias que tiene que hacer un árbol de decisión para encontrar un punto atípico (izquierda) vs. particiones necesarias para encontrar un punto normal (derecha).

Como en otros métodos de detección de puntos anómalos, en IF se requiere calcular una puntuación de anomalía para la toma de decisiones. Esta se calcula en base a todos los árboles y a la profundidad que alcanza la observación en cada uno. Una puntuación cercana a 1 indica que la observación es un punto atípico, y una puntuación mucho menor que 0.5 indica una observación normal [37].

Sin embargo, IF tiene un problema. Para ilustrarlo, pongamos un ejemplo: dado un conjunto de datos con distribución normal, esperaríamos que la puntuación de anomalía vaya aumentando según nos alejamos del centro del *cluster* de observaciones. Sin embargo, como los cortes que forman los árboles de IF con sus particiones son exclusivamente horizontales

o verticales (valor aleatorio de atributo aleatorio), la acumulación de estos en los *clusters* de observaciones normales durante la fase de entrenamiento de IF, acaban por crear áreas en las que también habrá muchas divisiones aunque no haya muchas observaciones. Esto provoca que en estas zonas la puntuación que se da a las observaciones sea más baja de lo que debería. Podemos ver este fenómeno en la figura 5.3. De hecho, si contamos con a partir de dos *clusters* de observaciones normales, este comportamiento lleva a la generación de *clusters* fantasma, como se puede ver en la figura 5.4 [39].

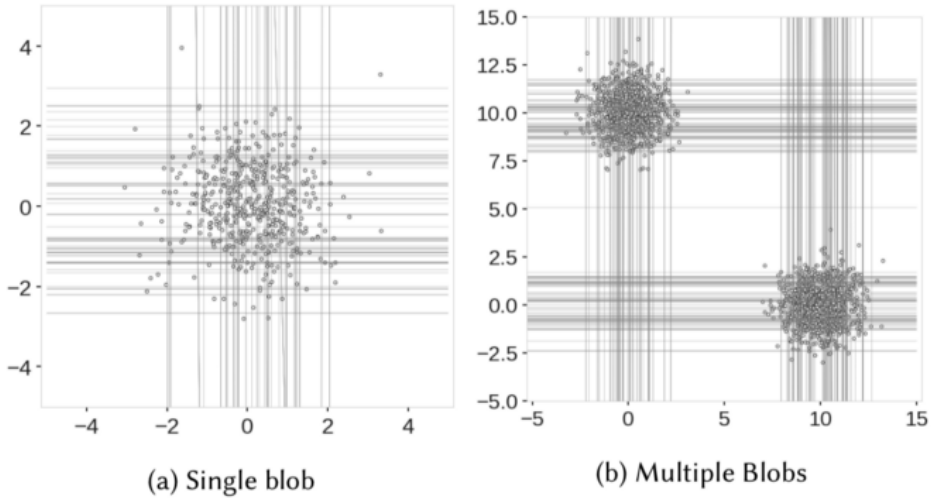


Figura 5.3: Particiones aleatorias necesarias generadas por IF durante el entrenamiento.

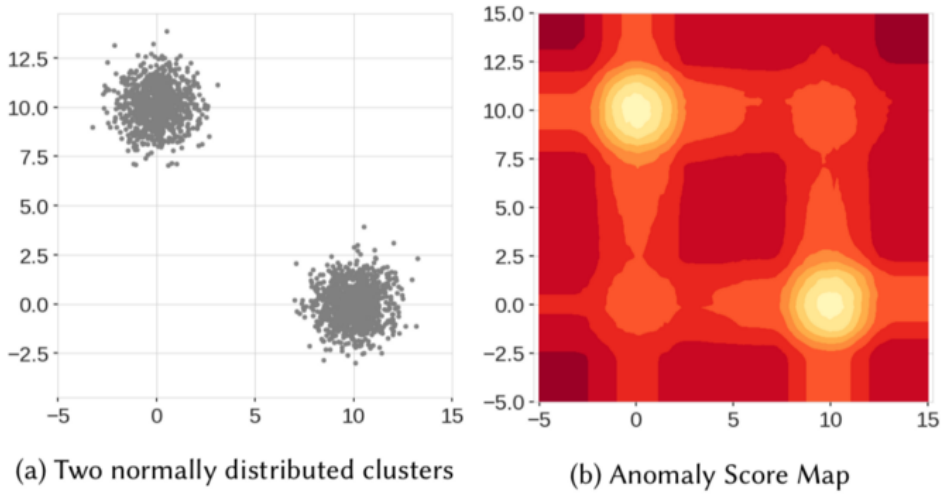


Figura 5.4: Mapa de puntuación de anomalías con dos *clusters* de observaciones normales (se crean otros dos *clusters* fantasma).

Existe una evolución de IF llamada Extended Isolation Forest (EIF), que arregla este problema empleando un acercamiento diferente [39]. No obstante, actualmente no está implementada en Scikit-learn. De hecho, parece que la razón es que la investigación sobre el algoritmo aún no tiene la consistencia suficiente [40]. En consecuencia, en este proyecto se emplea IF en vez de EIF para encontrar valores atípicos.

En lo que respecta a la aplicación de IF a nuestros datos, como se menciona al principio de la sección, lo empleamos sobre los atributos: *'totalSourceBytes'*, *'totalSourcePackets'*, *'totalDestinationBytes'* y *'totalDestinationPackets'*. Es decir, el algoritmo elimina las observaciones que sean anómalas en lo que respecta a estas cuatro variables. Aunque se puede establecer una indicación de cuántas anomalías creemos que hay en nuestro *dataset*, en este caso decidimos que el propio algoritmo lo determine en base a los datos. La aplicación del algoritmo resulta en la eliminación del 15% de muestras de nuestro *dataset*. Aunque parezca mucho, con un conjunto de datos de casi dos millones de observaciones, es una cantidad asumible. Respecto a la distribución de los atributos afectados tras la aplicación del algoritmo, podemos ver los diagramas de cajas y bigotes a continuación en la figura 5.5, figura 5.6, figura 5.7, y figura 5.8.

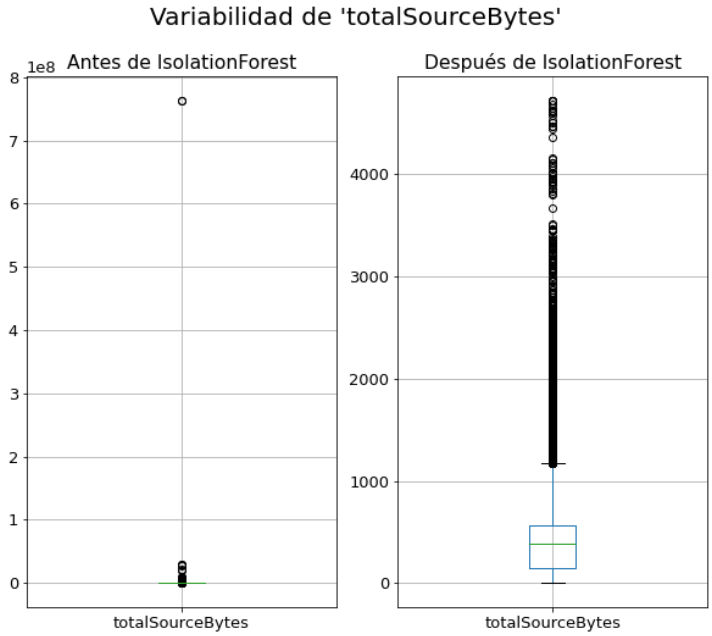


Figura 5.5: Representación de los datos en *'totalSourceBytes'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalSourceBytes'* después de aplicar IF (derecha).

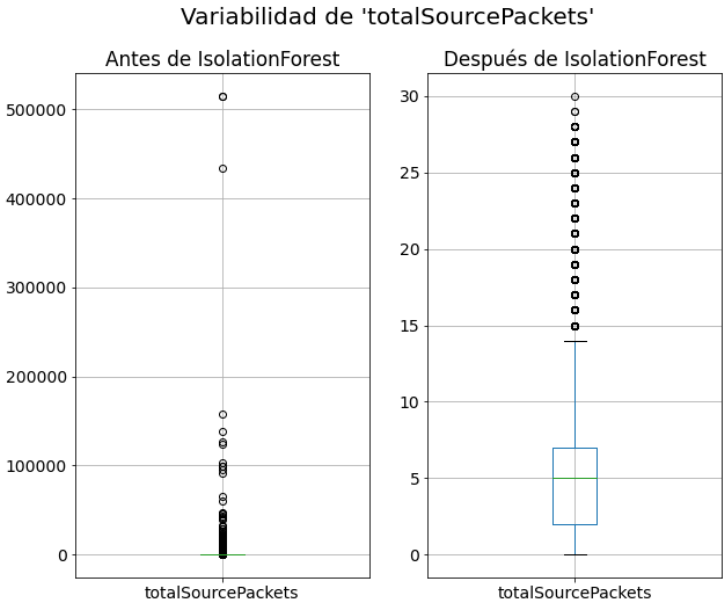


Figura 5.6: Representación de los datos en *'totalSourcePackets'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalSourcePackets'* después de aplicar IF (derecha).

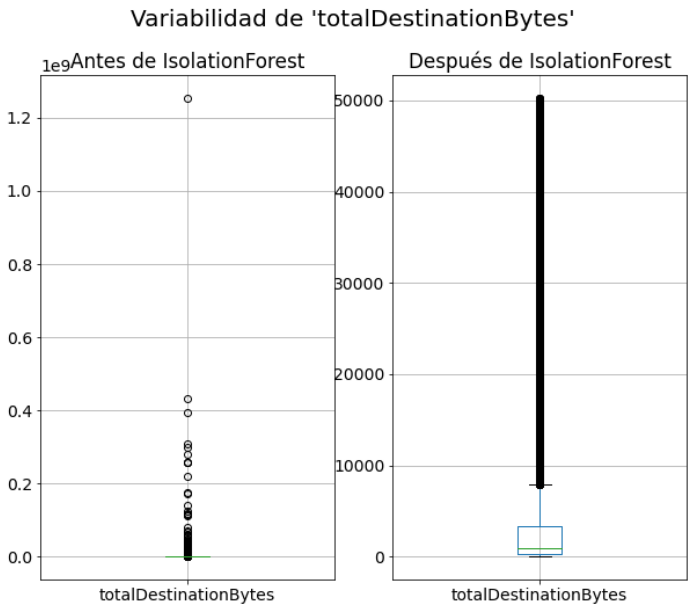


Figura 5.7: Representación de los datos en *'totalDestinationBytes'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalDestinationBytes'* después de aplicar IF (derecha).

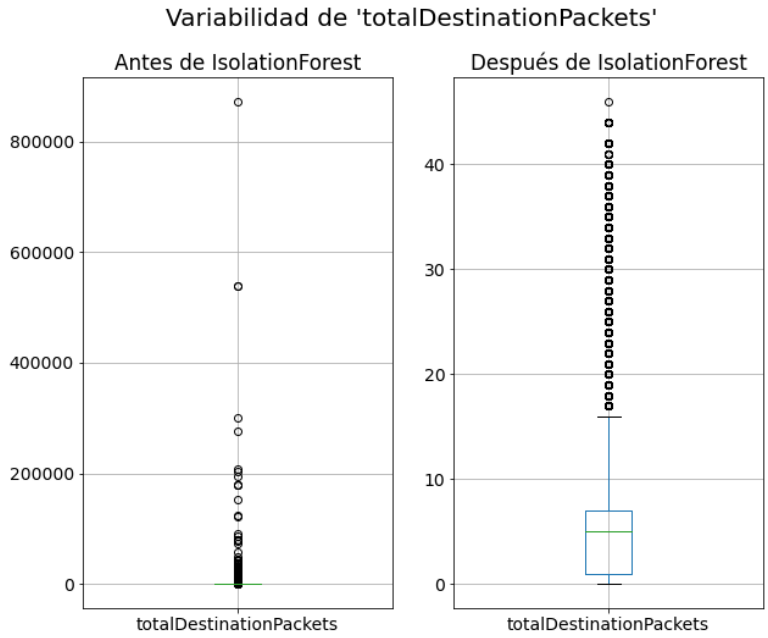


Figura 5.8: Representación de los datos en *'totalDestinationPackets'* antes de aplicar IF (izquierda) vs. representación de los datos en *'totalDestinationPackets'* después de aplicar IF (derecha).

5.2 Transformación

5.2.1 Cambio de formato

Los primeros atributos cuya representación cambiamos son *'sourceTCPFlagsDescription'* y *'destinationTCPFlagsDescription'*. Su formato original es una cadena de caracteres con la lista de *flags* separados por comas. Este formato es problemático, ya que incluso si se pudieran usar las cadenas de caracteres como etiquetas categóricas en el modelado, algo que Scikit-learn no permite, en el caso de que dos cadenas tuvieran los mismos *flags* en distinto orden, el algoritmo lo consideraría una categoría diferente. De entre las posibilidades de recodificación disponibles, se decide hacer uso de una variante de la técnica *One-Hot-Encoding* (OHE) [41]. Normalmente OHE crea por cada uno de los valores de la variable categórica a la que se aplica, una variable binaria. Cada una de estas nuevas variables indica si en la variable original aparecía la categoría que representa. En nuestro caso, en vez de crear una variable por cada lista de *flags* de la variable original, crearemos una variable binaria por cada *flag*, que indique si la lista del atributo original contiene ese *flag*. Elegimos aplicar OHE porque es una técnica especialmente adecuada para variables categóricas con poca cardinalidad, fácil de utilizar, y nos permite representar de manera más eficiente la información sobre los *flags*.

Hacemos algo similar con el atributo *'direction'*. Se trata de una variable categórica que puede tomar tres valores: *L2L*, *L2R* y *R2L*. Como Scikit-learn no soporta el uso de cadenas de caracteres como etiquetas categóricas, empleamos de nuevo OHE, esta vez en su versión estándar, para crear dos nuevas variables que recodifiquen la información de *'direction'*. Sólo creamos dos porque aunque la variable original puede tomar tres valores diferentes, sólo puede pertenecer a una de las categorías a la vez, así que al cambiar su representación con OHE por dos variables, que las dos estén a cero ya implica que se trata de la tercera categoría. Así pues, crear tres variables es mantener información redundante que puede afectar negativamente a nuestros resultados.

Con *'sourceTCPFlagsDescription'* y *'destinationTCPFlagsDescription'* no hay que prescindir de la variable de uno de los *flags* porque el hecho de que en una observación aparezca o no aparezca un *flag*, no tiene relación con que lo hagan los demás. Por lo tanto, si eliminamos uno de los atributos no estamos retirando información redundante, sino información válida.

También cambiamos el formato del atributo *'protocolName'*. Es una variable categórica de tan solo cinco valores posibles: *'tcp_ip'*, *'udp_ip'*, *'icmp_ip'* e *'igmp'*. Por la misma razón que en *'direction'*, aplicamos OHE para crear variables binarias que representen todos los protocolos menos uno. En este caso se elimina *'igmp'*.

Por último, modificamos la variable objetivo *'Target'*. En su formato original es una cadena de caracteres con los valores *'Normal'* para los flujos normales, y *'Attack'* para los de ataque. Como ya se ha comentado, Scikit-learn no soporta esa representación de variables categóricas, así que la convertimos en una variable binaria, asignando a los flujos de ataque el valor verdadero y a los normales el falso.

5.2.2 Modificación de atributos

Los atributos *'source'* y *'destination'*, como ya mencionábamos en la sección 4.2.2, son de naturaleza categórica y tienen una alta dimensionalidad. Contienen las IPs origen y destino de los flujos, codificadas como una cadena de caracteres. Consideramos que esta representación no es beneficiosa en ningún aspecto, ya que de ella los algoritmos no pueden extraer más información que cuáles son los flujos con la misma IP. Sin embargo, no son capaces de deducir que dos flujos provienen de la misma subred, una información mucho más interesante de cara a predecir la naturaleza de nuevas observaciones. Para nuestro objetivo de detectar anomalías en la red, que una IP tenga mucha incidencia en ataques no es relevante, porque identifica a una máquina que podría cambiar en cualquier momento. No obstante, saber que desde una subred provienen muchos ataques, o que muchos se dirigen hacia ella, sí lo es, porque nos da una localización que no va a cambiar y que es especialmente vulnerable. Por esa razón, decidimos transformar el atributo original para que exhiba mejor la información que nos interesa. Lo primero que hacemos es eliminar las IPs individuales y agruparlas se-

gún su subred, obteniendo las mismas categorías que cuando lo hicimos en la sección 4.2.2 para visualizar con más facilidad sus propiedades, a saber: *'192.168.1'*, *'192.168.2'*, *'192.168.3'*, *'192.168.4'*, *'192.168.5'*, *'192.168.6'*, *'external'* y *'others'*. A continuación, para que Scikit-learn pueda aprovecharse de la información contenida en la variable transformada, aplicamos de nuevo OHE. Como las categorías vuelven a ser exclusivas, tenemos que eliminar una variable de cada grupo de subredes (una de las subredes de *'source'* y una de las de *'destination*). En este caso se prescinde de las variables que representan la subred *'external'*, concretamente *'Sexternal'* y *'Dexternal'*.

Por otro lado, los atributos *'startDateTime'* y *'stopDateTime'* son las marcas temporales del comienzo y fin de cada flujo. Están formateadas como cadenas de caracteres, y tienen una precisión que va hasta los segundos. Como en el caso de *'source'* y *'destination'*, no es suficiente con pasarlas a un formato reconocible por Scikit-learn, porque de por sí las marcas temporales tampoco aportan demasiada información. En cambio, si agrupamos por la hora del día en la que cada flujo se ha enviado, obtenemos las horas en las que se originan más ataques, una información más interesante para modelar. Pese a todo, este acercamiento sigue siendo imperfecto. Las horas del día tienen un carácter cíclico, que los algoritmos no van a comprender si simplemente pasamos la hora como un número del 0 al 23. Es necesaria una transformación más. Concretamente, lo que vamos a hacer es mapear los valores a una circunferencia, de manera que el valor más pequeño aparezca al lado del más grande. Para hacerlo empleamos el seno y el coseno, que proyectan las horas del día en una circunferencia goniométrica, como la usada para estudiar las razones y funciones trigonométricas. Tenemos una representación visual de esta idea en la figura 5.9 [42] [43].

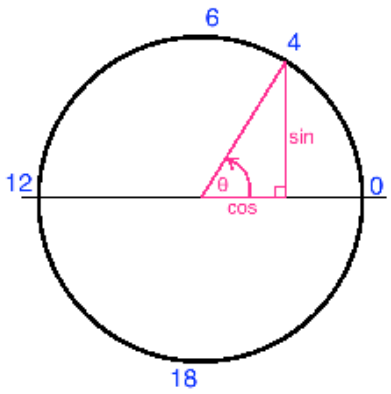


Figura 5.9: Circunferencia goniométrica con las horas del día.

5.2.3 Creación de atributos

Los dos primeros atributos que creamos son *'ratioSourceBytes'* y *'ratioDestinationBytes'*, a partir de *'totalSourceBytes'* y *'totalDestinationBytes'*. El procedimiento es el evidente, por cada observación, dividimos el valor de cada uno de los atributos entre la suma de ambos. Lo que reflejan los atributos generados es la proporción de información que se envía respecto a la que se recibe.

A continuación generamos *'sourceBitrate'* y *'destinationBitrate'*. Estos provienen de *'totalSourceBytes'*, *'totalDestinationBytes'*, *'startDateTime'* y *'stopDateTime'*. En primer lugar obtenemos la duración de los flujos, en segundos, restando a la marca temporal *'stopDateTime'* su correspondiente *'startDateTime'*. Como hay flujos que expiran tan rápido que no llegan a un segundo de duración, cambiamos todos los 0 en el atributo temporal *'duracionSeg'* por 0.5, de manera que en el siguiente paso no se den casos de división entre cero. Por último generamos los nuevos atributos con la expresión:

$$sourceBitrate = (totalSourceBytes \times 8)/duracionSeg$$

$$destinationBitrate = (totalDestinationBytes \times 8)/duracionSeg$$

En cuanto a *'sourcePacketsRate'* y *'destinationPacketsRate'*, estos proceden de *'totalSourcePackets'*, *'totalDestinationPackets'*, *'startDateTime'* y *'stopDateTime'*, Se obtienen mediante el mismo procedimiento que se emplea para *'sourceBitrate'* y *'destinationBitrate'*, pero sin tener que multiplicar por 8 en la división, ya que sólo era necesario para convertir los bytes en bits.

Por último, generamos *'sourcePacketSize'* y *'destinationPacketSize'*. Provienen de *'totalSourceBytes'*, *'totalDestinationBytes'*, *'totalSourcePackets* y *'totalDestinationPackets'*. Para crearlos empleamos las siguientes expresiones:

$$sourcePacketSize = totalSourceBytes/totalSourcePackets$$

$$destinationPacketSize = totalDestinationBytes/totalDestinationPackets$$

5.2.4 Escalado

Algunas técnicas de ML, como las redes neuronales o SVM, son muy sensibles al escalado de los datos. Esto se debe a que los algoritmos de ML no tienen la capacidad de saber cuál es el significado que los humanos les damos a los atributos, sólo ven números. Si existe una gran diferencia en el rango de valores de las diferentes variables, pueden llegar a asumir que los atributos con valores más altos son, de alguna manera, más relevantes [44].

El escalado nos ayuda a evitar que se le de más importancia a un atributo sólo por el rango de sus valores. Además, algunos algoritmos se benefician de maneras adicionales, por

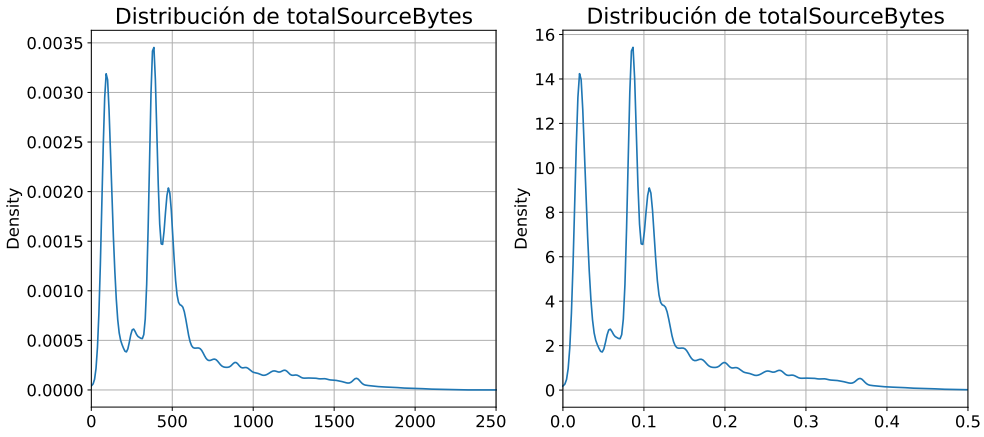
ejemplo, convergiendo más rápido [44].

Las técnicas que no requieren normalización o escalado son las que se basan en reglas, y por lo tanto no les afectan las transformaciones de las variables. Un ejemplo de estas es RF. Además, existen algoritmos como NB, que aunque no se basen en reglas, están equipados para lidiar con este problema dándoles pesos a las variables [44].

Hay diferentes formas de escalar datos. Entre ellas: *'Standard Scaler'*, *'Min Max Scaler'*, *'Robust Scaler'* y *'Normalizer'*. Cada una tiene sus ventajas e inconvenientes. Nosotros decidimos emplear *'Min Max Scaler'*, porque es una buena opción teniendo en cuenta que ya hemos eliminado los valores atípicos del conjunto previamente. Si no lo hubiéramos hecho, la mejor opción habría sido *'Robust Scaler'* [25].

'Min Max Scaler' transforma los atributos reescalándolos a un rango concreto de valores. Lo hace de manera que cada variable mantenga su distribución. En nuestro caso el rango de valores de los atributos reescalados es $[0, 5]$. Podemos ver un ejemplo de los resultados de esta transformación en la figura 5.10.

Como ya hemos comentado, el escalado de datos no se considera útil para RF o NB. Por lo tanto, en este proyecto sólo se empleará cuando trabajemos con DNN.



(a) Distribución de 'totalSourceBytes' antes del escalado con *Min Max Scaler*.

(b) Distribución de 'totalSourceBytes' después del escalado con *Min Max Scaler*.

Figura 5.10: Efectos del reescalado en *'totalSourceBytes'*. En (a), la distribución del atributo antes de ser reescalado. En (b), la distribución después.

5.3 Selección

El propósito principal de esta sección es elegir los datos que emplearemos para modelar. Hay varios aspectos a tener en cuenta, como lo relevantes que son los atributos para alcanzar nuestros objetivos, las restricciones y características de los algoritmos que vamos a emplear

para el modelado, o qué métodos usar tanto para la selección de muestras, como de atributos. En la sección 5.3.1 y la sección 5.3.2 profundizaremos en las soluciones empleadas en este trabajo.

5.3.1 Selección de muestras

Las técnicas de ML suelen dar por hecho que los conjuntos de entrenamiento tienen una representación equilibrada de eventos. Sin embargo, este no es siempre el caso en los *datasets* reales, donde una clase puede estar representada por un gran número de observaciones, mientras que otra sólo por unas pocas. Cuando esto ocurre decimos que tenemos un problema de desbalanceo de clases en nuestro *dataset*, porque normalmente supone un obstáculo para la creación de buenos clasificadores [45].

Existen muchas técnicas para abordar esta situación como el sobremuestreo o *oversampling*, la estratificación o *stratification*, y el submuestreo o *undersampling*. En este proyecto y teniendo en cuenta las características de los datos, decidimos usar submuestreo aleatorio o *random undersampling*. Consiste en la retirada aleatoria de muestras de la clase dominante. La elección de este método se debe a que tenemos un número suficiente de observaciones de flujos de ataque, y una cantidad desorbitada de muestras de flujos normales. La librería *imbalanced-learn* cuenta con una implementación muy sencilla de utilizar, que permite la introducción de una semilla para obtener resultados replicables [46]. En la figura 5.11 podemos ver el resultado de aplicar el submuestreo a nuestro *dataset*. Pasamos de tener más de 1.5 millones de observaciones de flujos normales, a 30 mil, alcanzando finalmente un equilibrio con el número de flujos de ataque.

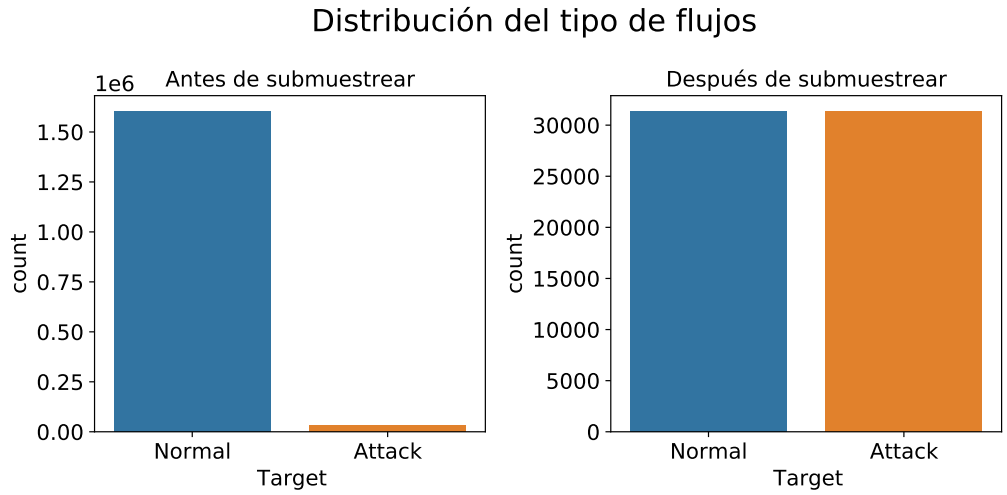


Figura 5.11: Efecto de aplicar *random undersampling* al conjunto de datos.

5.3.2 Selección de atributos

El número de atributos en nuestros *datasets* se ha expandido en los últimos años, pasando de decenas a cientos. En consecuencia, han sido desarrollados varios métodos para reducir variables redundantes o irrelevantes, que pueden suponer una carga para las tareas de ML [47].

La eliminación de atributos no útiles propicia la extracción de conocimiento a partir de los datos, la reducción de los requisitos computacionales, la reducción de los efectos de la *maldición de la dimensionalidad*, y la mejora de la eficacia de los clasificadores. El objetivo es, por lo tanto, escoger un subconjunto de variables de entrada que pueda describir eficientemente los datos, al mismo tiempo que reducimos los efectos del ruido, y mantenemos los buenos resultados de predicción [47].

En este proyecto la selección de atributos se basa en el conocimiento adquirido en las fases anteriores, tanto sobre los datos como sobre las técnicas de ML. Esto sólo es posible porque el número de atributos del *dataset* que empleamos es reducido, y por lo tanto abarcable manualmente. Aunque esta situación parezca contradictoria con el aumento en la cantidad de atributos mencionado al principio del apartado, cabe recordar que el *dataset* que estamos empleando es el UNB ISCXIDS2012, y que se remonta a 2012. El UNB CIC IDS 2017, otro conjunto del mismo dominio pero más reciente, alcanza más de 80 atributos [48].

Por último, como uno de los objetivos principales de este trabajo es encontrar la técnica que mejor se adapte a nuestro problema, se intentará mantener la lista de variables que empleamos, uniforme entre las diferentes técnicas. Lo hacemos así porque si cambiáramos demasiado el conjunto de un algoritmo a otro, sería complicado hacer una comparativa válida. Se considerará el conjunto base de atributos a todos los atributos generados en la sección 5.2.1, sección 5.2.2, y sección 5.2.3, menos *'L2L*, *'L2R'*, y *'R2L'*, porque la información de estos últimos está contenida en los atributos de las subredes. La descripción de la distribución de los atributos está disponible en el anexo A.3.

Random Forest

RF es, de los tres algoritmos que se utilizan en este trabajo, el que menos restricciones tiene. Acepta atributos con cualquier distribución, de naturaleza categórica o continua, y no necesita que los datos estén escalados. Para RF empleamos los atributos del conjunto base sin escalado.

Naïve Bayes

En la sección 3.2.1 comentamos que NB tiene 3 variantes, cada una de ellas especializada en atributos con una distribución en concreto. Como no existe ninguna variante adecuada para

todos nuestros atributos, decidimos emplear un enfoque híbrido. Para ello primero dividimos el conjunto de variables, separando las que tienen una distribución binaria de las continuas. A continuación, usamos las del primer grupo para entrenar un modelo Bernoulli NB, y las del segundo para un modelo Gaussian NB. Lo que hacemos con esto es crear dos modelos intermedios, que intentan predecir la naturaleza del flujo en base a la parte de los atributos con una distribución afín a sus asunciones. Por último, juntamos las probabilidades de pertenencia a cada clase, que cada uno de los dos modelos intermedios generan, y empleamos estas probabilidades como nuevo *dataset* para generar el modelo Gaussian NB definitivo.

Con este acercamiento conseguimos emplear todos los atributos de nuestro *dataset* en NB, sin tener que renunciar a usar, para cada algoritmo, la distribución en la que se especializa.

Deep Neural Network

Por último, para DNN empleamos también todas las variables del conjunto base. Lo único que cambia respecto a RF, es que en esta ocasión las escalamos primero. En la sección 5.2.4 hablamos de las razones por las que el escalado es tan importante para algunas técnicas de ML y DL. Las redes neuronales se encuentran entre estas técnicas, suponiendo para ellas un paso crítico con notorios efectos en su rendimiento.

Capítulo 6

Modelado

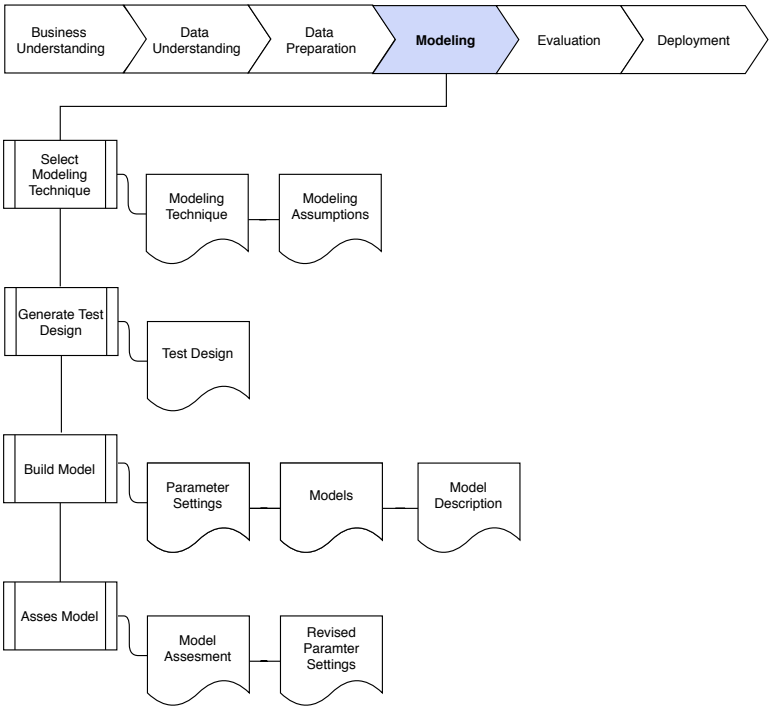


Figura 6.1: Tareas de la fase cuatro de CRISP-DM: modelado.

En este capítulo hablaremos de la cuarta fase de CRISP-DM. La figura 6.1 nos muestra las tareas que la conforman:

- Selección de técnicas de modelado: en esta tarea, en caso de no haberlo hecho, deberíamos elegir los algoritmos que vamos a usar para el modelado. En este caso, ya están determinados y explicados en la sección 3.2.
- Diseño de pruebas: en esta tarea establecemos los mecanismos que utilizamos para medir la calidad y validez de los modelos. Esto se hace en la sección 6.1.

- Construcción de modelos: esta tarea gira alrededor de la creación de los modelos a partir de los algoritmos que hemos seleccionado. Se aborda conjuntamente con la siguiente tarea en la sección 6.2.

- Validación del modelo: en esta tarea se deberán interpretar los modelos y el éxito de los mismos teniendo en cuenta los resultados de las pruebas, el conocimiento del dominio, y los criterios de éxito establecidos. A diferencia de la fase de evaluación, esta actividad se centra en los resultados de los modelos, en vez de en los resultados generales del proyecto. Acometemos esta tarea en la sección 6.2.

6.1 Evaluación de modelos

6.1.1 *Cross-Validation*

La validación cruzada o *cross-validation*, es un método estadístico de evaluación del rendimiento en modelos, más escalable y exhaustivo que simplemente dividir un conjunto de datos en subconjuntos de entrenamiento y test. La versión de *cross-validation* más comúnmente usada es *k-fold cross-validation*, donde k es un número especificado por el usuario, normalmente 5 o 10.

El proceso de aplicar validación cruzada con cinco particiones consiste en los siguientes pasos: primero los datos se dividen en cinco particiones del mismo tamaño. A continuación, se crea una secuencia de modelos. El primero se entrena empleando la primera partición como conjunto de test, y las restantes como conjunto de entrenamiento. Después se construye otro modelo, pero esta vez empleando la partición dos como conjunto de test, y las particiones uno, tres, cuatro, y cinco como conjunto de entrenamiento. Este proceso se repite para todas las particiones. Para cada una de las configuraciones, calculamos la precisión. Al final, obtenemos cinco precisiones diferentes, a las que podemos hacer la media para saber cuál es el comportamiento general del algoritmo en esos datos.

Cabe aclarar que *cross-validation* no es una manera de construir un modelo que pueda ser aplicado a nuevos datos. Cuando ejecutamos *cross-validation* se construyen varios modelos de manera interna, pero el propósito es evaluar cómo de bien puede generalizar un algoritmo cuando es entrenado con un *dataset* específico.

Por último, dividir el conjunto de datos en particiones sin más comprobaciones, puede no ser una buena idea en todos los casos. Si por alguna razón las observaciones de conjunto de datos están ordenados por clase, coger una porción de los datos puede suponer coger sólo observaciones de una clase, lo que provocaría malos resultados. Para tareas de clasificación se debería emplear la variante de *cross-validation* llamada validación cruzada estratificada, o *stratified k-fold cross-validation*. En esta, antes de crear las particiones de los datos, nos ase-

guramos de que cada partición tenga la misma proporción de clases que el conjunto original.

En la figura 6.2, podemos ver la diferencia entre *stratified k-fold cross-validation* y *k-fold cross-validation*.

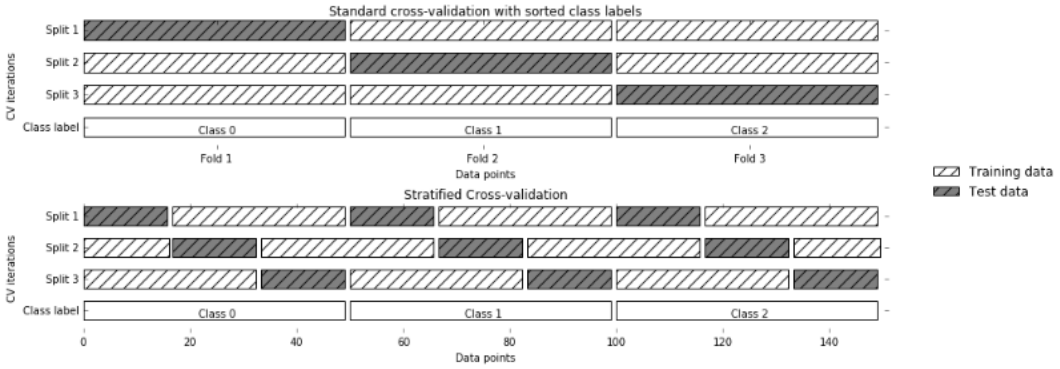


Figura 6.2: Diferencias entre la versión estándar de *cross-validation* (arriba) y la versión estratificada (abajo).

6.1.2 Métricas de evaluación

Antes de describir las diferentes métricas con las que contamos para valorar el rendimiento predictivo de nuestro modelo, tenemos que definir algunos conceptos. Pensemos en una tarea de clasificación binaria, en la que establecemos una de las clases como *positiva*, y otra como *negativa*. Entonces, cada vez que nuestro modelo prediga una observación, podemos tener un [19]:

- Verdadero positivo o *true positive* (TP): es una observación de la clase positiva que clasificamos como positiva.
- Falso positivo o *false positive* (FP): es una observación de la clase negativa que clasificamos como positiva.
- Verdadero negativo o *true negative* (TN): es una clasificación de la clase negativa que clasificamos como negativa.
- Falso negativo o *false negative* (FN): es una clasificación de la clase positiva que clasificamos como negativa.

Estas ideas sirven de fundamento para lo que llamamos matriz de confusión. Es una de las maneras más completas de representar el rendimiento de un modelo que realiza clasificación binaria. En la figura 6.3, podemos apreciar cómo son [19].

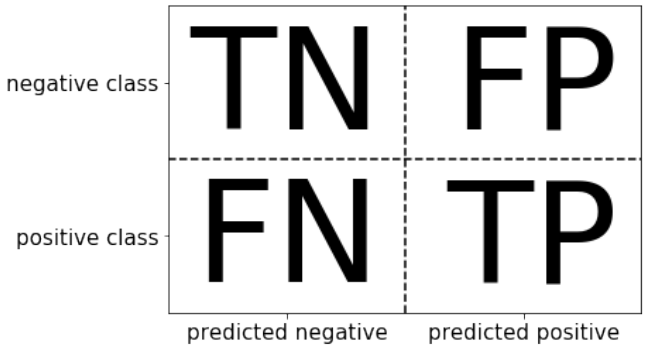


Figura 6.3: Matriz de confusión para clasificación binaria.

Existen varias maneras de resumir la información que nos da una matriz de confusión. Algunas de las más importantes son:

- Precisión o *accuracy*: representa el número de predicciones correctas dividido entre el número de todas las muestras.

$$Precision = \frac{TP + TN}{TP + TN + FP + FN}$$

- Exactitud o *precision*: representa cuántas de las muestras que se han clasificado positivas son realmente positivas. Se emplea cuando el objetivo es limitar los falsos positivos.

$$Exactitud = \frac{TP}{TP + FP}$$

- Sensibilidad o *recall*: representa la cantidad de muestras positivas que han sido clasificadas como positivas. Se emplea cuando es importante evitar falsos negativos.

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

Existe un equilibrio entre la optimización de la sensibilidad y la exactitud. Si lo pensamos, es trivial obtener una sensibilidad perfecta si ignoramos la exactitud, sólo tendríamos que predecir todas las observaciones como pertenecientes a la clase positiva. Sin embargo, esto resultaría en una alto número de *falsos positivos*, y por lo tanto la exactitud sería muy baja. Por el otro lado, ocurriría lo contrario si sólo predecimos como perteneciente a la clase positiva a la observación de la que más seguros estemos, y el resto las predecimos como negativas [19].

En consecuencia, es necesario que tengamos en cuenta ambas métricas de manera conjunta para entender realmente cómo está funcionando nuestro modelo. Una manera de resumir la información de las dos es la *f-score*, la media armónica de la sensibilidad y la exactitud [19]:

$$F = \frac{sensibilidad \cdot exactitud}{sensibilidad + exactitud}$$

Esta variante en particular también se conoce como $f_1 - score$. Como tiene en cuenta tanto la exactitud como la sensibilidad, puede ser una mejor medida que la precisión en conjuntos desbalanceados [19].

Teniendo en cuenta que nuestro principal objetivo e interés es que las amenazas en nuestra red no permanezcan ocultas, una métrica a la que habrá que prestar especial atención es la sensibilidad.

6.2 Construcción del modelo e hiperparametrización

Los modelos de ML están parametrizados, de manera que su comportamiento pueda ser adaptado a problemas concretos. Encontrar los valores de los parámetros que provean la mayor capacidad de generalización posible es una tarea complicada, pero necesaria para casi todos los modelos y *datasets*. En esta sección detallamos los parámetros de los tres algoritmos seleccionados para este proyecto, y buscamos los mejores valores para ellos.

La búsqueda consistirá en la creación y comparación de modelos con diferentes combinaciones de parámetros. Como es una tarea muy habitual, existen métodos en Scikit-learn para ayudarnos a llevarla a cabo. El más común es *grid search* [49], una herramienta que nos facilita probar todas las combinaciones posibles de las listas de parámetros que le pasamos. Además, para asegurarnos de que la combinación ganadora es realmente la mejor, integramos *grid search* con *stratified k-fold cross-validation* con 3 particiones, evitando así que los resultados se vean afectados por una división fortuita de los datos. Cuando tengamos la mejor configuración para nuestros modelos, los probaremos con el conjunto de test. Como vamos a necesitar entrenar 3 veces por cada combinación de parámetros, se acotan los rangos de los mismos para paliar el coste temporal.

6.2.1 *Random Forest*

Los parámetros más importantes a configurar en RF son:

- Número máximo de atributos por partición: hace referencia al número de características que se pueden utilizar en cada partición. En Scikit-learn se controla con el parámetro *'max_features'*.

- Número de árboles: hace referencia al número de árboles de decisión que conforman el RF. En Scikit-learn se controla con el parámetro *'n_estimators'*.

Podemos observar en la figura 6.4 las diferentes combinaciones los parámetros probadas. Cabe aclarar que *'auto'* establece como valor de *'max_features'* la raíz cuadrada del número de atributos, así que en este caso equivale a 7.

Los mejores valores para nuestro modelo resultan ser 20 para *'max_features'* y 400 para *'n_estimators'*. Respecto a los resultados que se obtienen con esta configuración, su matriz de confusión se encuentra en la figura 6.5, y sus métricas calculadas en la figura 6.6.

```
GridSearchCV(cv=3, estimator=RandomForestClassifier(),
             param_grid={'max_features': ['auto', 2, 3, 5, 10, 20],
                         'n_estimators': [300, 400, 700, 1000, 1500, 2000]},
             verbose=1)
```

Figura 6.4: Configuración de la red de búsqueda de parámetros para RF.

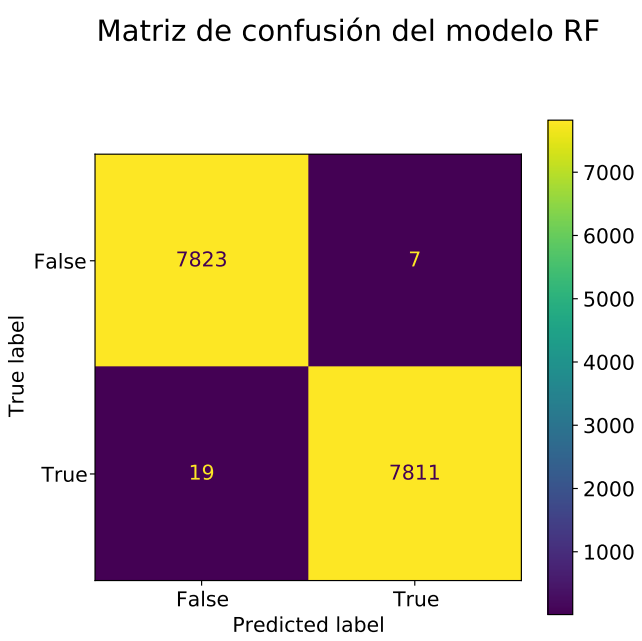


Figura 6.5: Matriz de confusión del modelo RF con mejores parámetros sobre el conjunto de test.

Los resultados de RF son excepcionalmente positivos. De hecho, la razón por la cual la etapa del proyecto en la que nos centramos en esta técnica se alargó, es que su rendimiento era sorprendente incluso con escasos atributos sin apenas transformar. La conclusión a la que se llegó es que en el *dataset* hay un grupo reducido de atributos que perfilan extremadamente bien el tipo de flujo, sobre todo a nivel diario. De todos modos, no descartamos que la proce-

dencia artificial del conjunto de datos tenga un papel en este fenómeno. La importancia que da RF a cada uno de los atributos en la predicción está disponible en el anexo A.4.

	precision	recall	f1-score	support
Normal	0.99758	0.99911	0.99834	7830
Attack	0.99910	0.99757	0.99834	7830
accuracy			0.99834	15660
macro avg	0.99834	0.99834	0.99834	15660
weighted avg	0.99834	0.99834	0.99834	15660

Figura 6.6: Métricas del rendimiento del modelo RF con mejores parámetros sobre el conjunto de test.

6.2.2 *Naïve Bayes*

Aunque usamos dos variantes de NB, sólo tenemos que optimizar un parámetro de Bernoulli NB:

- Alpha: controla la complejidad del modelo. El algoritmo añade *alpha* puntos virtuales con valores positivos para todos los atributos. Esto provoca una suavización de las estadísticas. Un valor alto de este parámetro supone una mayor suavización, lo que induce a modelos más complejos.

En la figura 6.7 mostramos los valores de *alpha* que probamos para el modelo Bernoulli NB. Por otro lado, en la figura 6.9 se encuentra la matriz de confusión del modelo Gaussian NB final, y en figura 6.8 las estadísticas derivadas. El mejor valor de *alpha* en este caso es 1×10^{-5}.

```
GridSearchCV(cv=3, estimator=BernoulliNB(),
             param_grid={'alpha': [1e-05, 0.0001, 0.001, 0.01, 0, 1.0, 10.0,
                                   100.0, 1000.0]},
             verbose=1)
```

Figura 6.7: Configuración de la red de búsqueda de parámetros para Bernoulli NB.

	precision	recall	f1-score	support
Normal	0.81237	0.96437	0.88187	7830
Attack	0.95617	0.77727	0.85749	7830
accuracy			0.87082	15660
macro avg	0.88427	0.87082	0.86968	15660
weighted avg	0.88427	0.87082	0.86968	15660

Figura 6.8: Métricas del rendimiento del modelo final Gaussian NB con mejores parámetros sobre el conjunto de test.

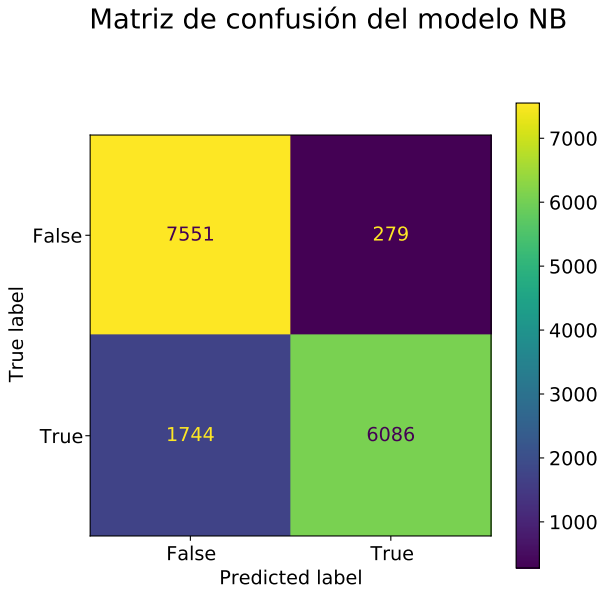


Figura 6.9: Matriz de confusión del modelo final Gaussian NB con mejores parámetros sobre el conjunto de test.

En general, consideramos que los resultados que obtiene NB no son nefastos, pero tampoco buenos, especialmente la sensibilidad para los flujos de ataque.

Por otro lado, encontramos relevante comentar una de las pruebas realizadas, ya que sienta las bases de nuestra interpretación sobre por qué no tenemos un rendimiento mejor: haciendo experimentos sobre el comportamiento de las técnicas NB con diferentes combinaciones de atributos, decidimos intentar introducir las variables *'L2L'*, *'L2R'* y *'R2L'*. En la sección 5.3.2 explicamos que se prescinde de estos atributos para el grupo final porque su información está incluida en las variables binarias de las subredes, así que esencialmente estábamos probando cómo reacciona NB a la presencia de variables correlacionadas. Sorprendentemente, Bernoulli NB, el modelo intermedio donde *'L2L'*, *'L2R'* , y *'RL2'* tenían que ser empleadas, alcanzó un 97% de precisión, respecto al 86.9% que alcanza con el grupo final de atributos.

Nuestra hipótesis es que NB es demasiado simple como para dar una importancia mayor a los atributos de las subredes, que contienen la información de dirección del flujo. Sin embargo, cuando introducimos otra fuente de la misma información, al tener que considerar ambas, creamos un sesgo que resulta aparentemente beneficioso.

Por último, después de ver los resultados, no consideramos que merezca la pena, en este caso, emplear conjuntamente Gaussian NB y Bernoulli NB. La razón es que la diferencia entre la precisión del modelo conjunto final, y sólo Bernoullin NB, es de tan sólo de un 0.0016%.

6.2.3 *Deep Neural Network*

DNN es, de entre los que estamos evaluando, el algoritmo para el que resulta más crítica la búsqueda de parámetros. Trabajamos con [50][51]:

- Tamaño de lote o *'batch size'*: es el número de observaciones que ponemos a disposición del modelo antes de que se produzca una actualización de los pesos de las conexiones.
- Repeticiones o *'epochs'*: son las veces que le enseñamos el *dataset* entero al modelo durante el entrenamiento.
- Algoritmo de optimización o *'optimization algorithm'*: es la función que ayuda a minimizar el valor del error de salida mediante la modificación de los pesos de las conexiones. Aunque se puede tratar como un parámetro más que probar, es común que se elija de antemano un algoritmo de optimización, y se trabaje en buscar los mejores parámetros para este. En este caso, en aras de acortar el proceso, se elige el algoritmo *'Adam'* y se mantienen sus parámetros por defecto.
- Inicialización de pesos de la red: es la técnica que se emplea para poner el valor inicial a los pesos de las conexiones.
- Función de activación o *'activation function'*: es la función que incorpora la suma ponderada de todas las entradas de la capa anterior, y genera un valor de resultado que pasa a la siguiente capa.
- Regularización de retirados o *'dropout regularization'*: es la proporción de nodos que retiramos durante el entrenamiento para evitar el sobreajuste.
- Número de neuronas de las capas ocultas: las neuronas son los nodos que hay en cada capa de la red.
- Número de capas ocultas: es el número de capas de la red.

Para los valores iniciales, se toman como referencia los de en un trabajo similar al nuestro [20]. Los parámetros finales y la arquitectura de la red están disponibles en el anexo A.5. Con ellos, la red neuronal alcanza los resultados que vemos reflejados en la matriz de confusión de la figura 6.10. Las métricas sobre la matriz de confusión están en la figura 6.11. Como se puede apreciar, los resultados de DNN son comparables a los de RF.

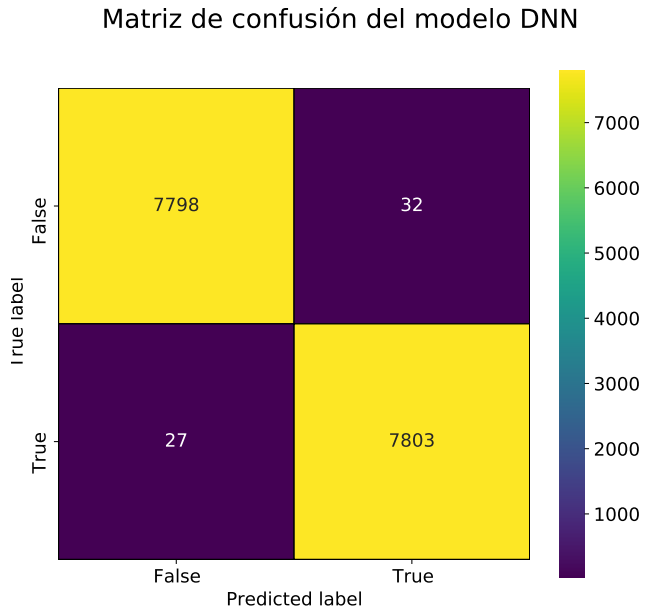


Figura 6.10: Matriz de confusión del modelo final DNN con mejores parámetros sobre el conjunto de test.

	precision	recall	f1-score	support
Normal	0.99655	0.99591	0.99623	7830
Attack	0.99592	0.99655	0.99623	7830
accuracy			0.99623	15660
macro avg	0.99623	0.99623	0.99623	15660
weighted avg	0.99623	0.99623	0.99623	15660

Figura 6.11: Métricas del rendimiento del modelo final DNN con mejores parámetros sobre el conjunto de test.

Capítulo 7

Evaluación

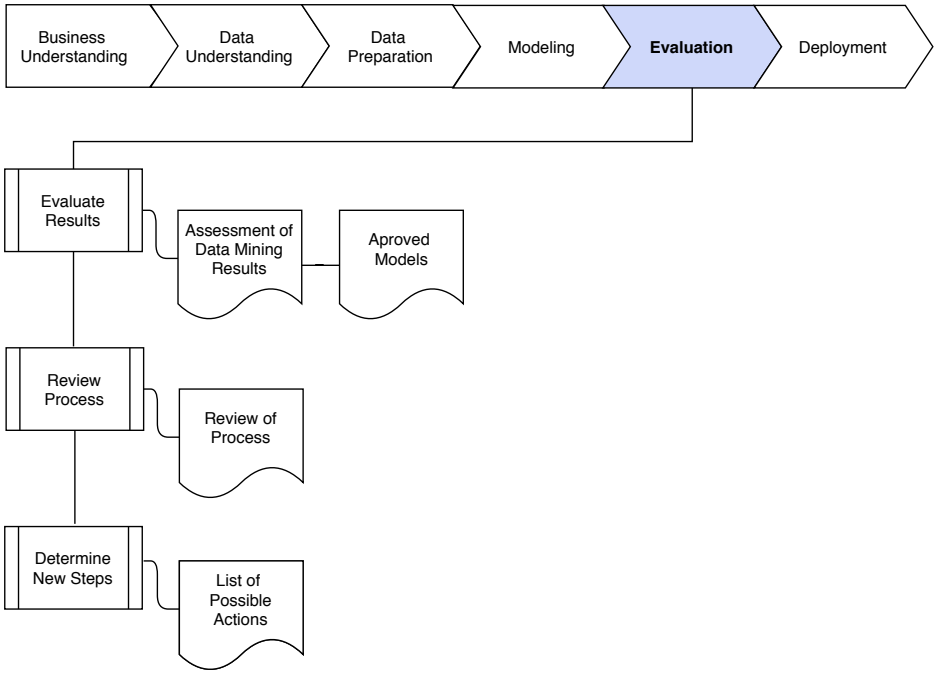


Figura 7.1: Tareas de la fase cinco de CRISP-DM: evaluación.

Este capítulo trata la penúltima fase del ciclo de vida CRISP-DM. En la figura 7.1 podemos ver sus actividades:

- Evaluación de resultados: en esta tarea debemos determinar el grado en el que los modelos generados cumplen los objetivos de negocio, así como discutir cualquier resultado relevante. Se lleva a cabo en la sección 7.1.
- Revisión del proceso: en esta tarea reflexionamos sobre el proceso que se llevó a cabo para la generación de modelos. Debemos indicar qué se puede mejorar, y cualquier

aspecto importante. Se llevará a cabo en el capítulo 9.

- Determinación de los siguientes pasos: en esta tarea, teniendo en cuenta los resultados de las tareas anteriores, se establecen los siguientes pasos a tomar. Se llevará a cabo en el capítulo 9.

7.1 Evaluación de resultados

Teniendo en cuenta las métricas sobre los resultados de los modelos en la figura 6.5, figura 6.9, y figura 6.10, generamos la tabla 7.1.

	Precision	**Recall**	**F1-Score**	**Accuracy**
RF	0,99910	0,99757	0,99834	0,99834
NB	0,95617	0,77727	0,85749	0,87082
DNN	0,99592	0,99655	0,99623	0,99623

Tabla 7.1: Métricas de cada modelo dando por hecho que la clase positiva es ser un flujo de ataque.

Como podemos apreciar, NB se queda bastante atrás con respecto a RF y DNN. No es algo extraño si tenemos en cuenta las características individuales de las técnicas. NB es un algoritmo sencillo y rápido, pero a cambo tiene problemas representando problemas excesivamente complejos. Esa es la razón de que a menudo se utilice como punto de partida o *baseline* de rendimiento para proyectos de ML [52] [53]. Por el otro lado, RF y DNN son algoritmos mucho más complejos, y en consecuencia costosos computacionalmente, pero también suelen aportar mejores resultados porque son capaces de llegar a donde NB no puede.

En lo que respecta a DNN contra RF, está bastante reñido. No se puede discutir que RF no sólo es mucho más fácil de parametrizar que DNN, sino que además es el que mejores resultados da. No obstante, consideramos que transformaciones más complejas de algunas variables, y una hiperparametrización más concienzuda, podrían potencialmente permitir a DNN sobrepasar a RF.

Sea como fuere, el algoritmo ganador de este proyecto es, en todos los aspectos, RF. En consecuencia, será este el que pasará a fase de despliegue para ser distribuido.

Capítulo 8

Despliegue

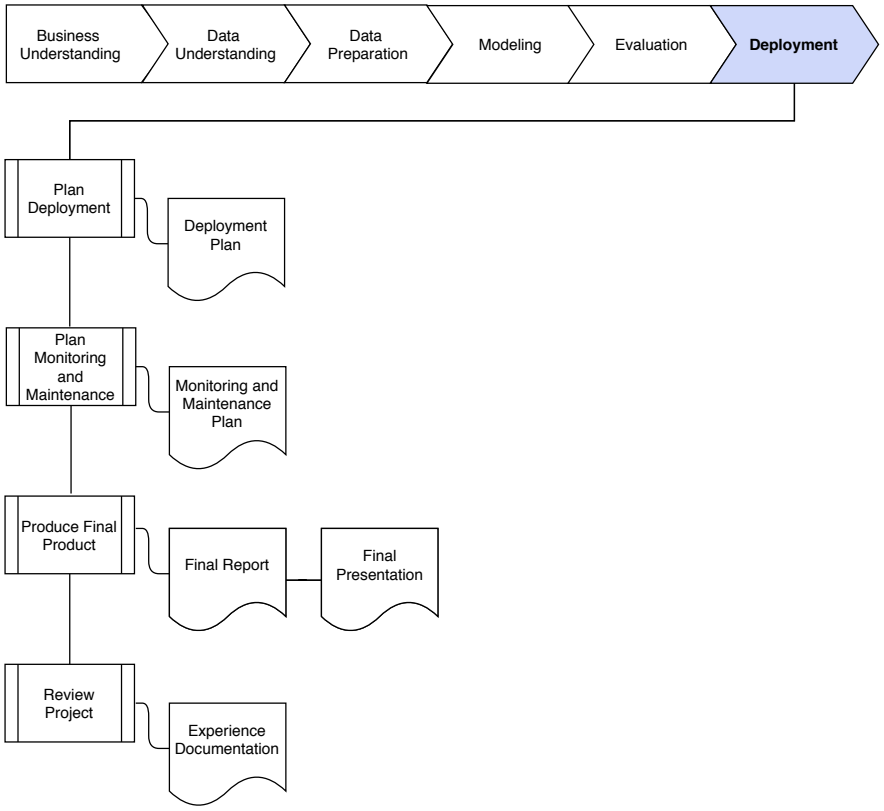


Figura 8.1: Tareas de la fase seis de CRISP-DM: despliegue.

En este capítulo hablamos de la última fase del ciclo de vida CRISP-DM. Las tareas que la componen están disponibles en la figura 8.1:

- Planificación del despliegue: en esta tarea usamos los resultados de la fase de evaluación para determinar una estrategia de despliegue. Se aborda en la sección 8.1.

- Monitorización y mantenimiento: en esta tarea trazamos las estrategias de monitorización y mantenimiento que vamos a emplear, para evitar problemas en el sistema desplegado. Se abordará en el capítulo 9.

- Generación del informe final: esta tarea consiste en escribir un informe final del proyecto que contenga un resumen del mismo, y las experiencias que ha supuesto. Se abordará en el capítulo 9.

- Revisión del proyecto: en esta tarea revisamos lo que fue bien, lo que fue mal, y lo que hay que mejorar en el proyecto. Se abordará en el capítulo 9.

8.1 Plan de despliegue

8.1.1 Creación del sistema

Como ya hemos mencionado antes, RF es el algoritmo que mejores resultados obtuvo en la fase de modelado. Por ello, también es el candidato a ser desplegado y distribuido.

Por otro lado, hasta ahora veníamos trabajando con Python y Scikit-learn para todas las etapas de nuestro proyecto. Sin embargo, en esta fase es necesario cambiarlos por PySpark y MLlib [54]. Como se explica en la sección 3.5, PySpark es una de las opciones que nos da Apache Spark Structured Streaming para trabajar en su plataforma. Respecto a MLlib, es una librería de ML perteneciente a Apache Spark, que permite la creación de modelos escalables.

El primer paso para el despliegue es implementar el modelo RF con MLlib, y guardar la versión distribuible. MLlib tiene herramientas que cubren todo el flujo de trabajo: transformación de variables, creación de *pipelines*, evaluación de modelos y persistencia. Si estuviéramos haciendo un sistema para un entorno real, tendríamos que crear, además del modelo, una *pipeline* de transformaciones que convirtieran la información que recibimos, posiblemente en un formato similar al del *.XML* de los archivos fuente del *dataset*, en el conjunto de datos preparado para que el modelo los procese. En aras de simplificar y acelerar el proceso, se da por hecho que los datos se envían como filas ya preprocesadas del conjunto.

Una vez tenemos el modelo listo, es necesario generar un *script* en PySpark que contenga los detalles del *streaming*, y del procesamiento que deseamos hacer sobre él. En este *script* también cargaremos el modelo previamente guardado, e indicaremos a Spark que según lleguen nuevas filas, han de ser procesadas por el mismo. Respecto a la salida con el resultado de la predicción, se ha optado por escribirla en un archivo, para poder leerlo a posteriori.

En cuanto al flujo de datos que se emplea para probar el sistema, está conformado por las filas del conjunto de test, enviadas de manera periódica con un bucle para imitar la llegada de tráfico en *streaming*.

También distribuimos el almacenamiento que empleamos para guardar el modelo, los esquemas de datos y los resultados. Lo hacemos con ayuda de Hadoop [55], creando un Hadoop Distributed File System (HDFS). Un HDFS es un sistema de ficheros distribuido, escalable y diseñado para ser tolerante a fallos, que nos da acceso a los datos guardados en el *cluster* que le da soporte.

Por último, empleamos dos acercamientos diferentes para evaluar la carga que le supone a Spark realizar tanto la ingesta de datos como el procesamiento del *stream*. Nuestra hipótesis es que si dejamos que Spark haga la ingesta de datos, puede generarse un sobrecoste computacional u *overhead* que merme los beneficios que nos aporta la distribución. Por esa razón comprobamos si Kafka [56] supone alguna diferencia como ingestador de datos en el segundo acercamiento.

Finalmente, en la figura 8.2 [57] podemos ver una representación bastante precisa de cómo intentamos integrar estas tres tecnologías para distribuir y desplegar nuestro modelo de ML.

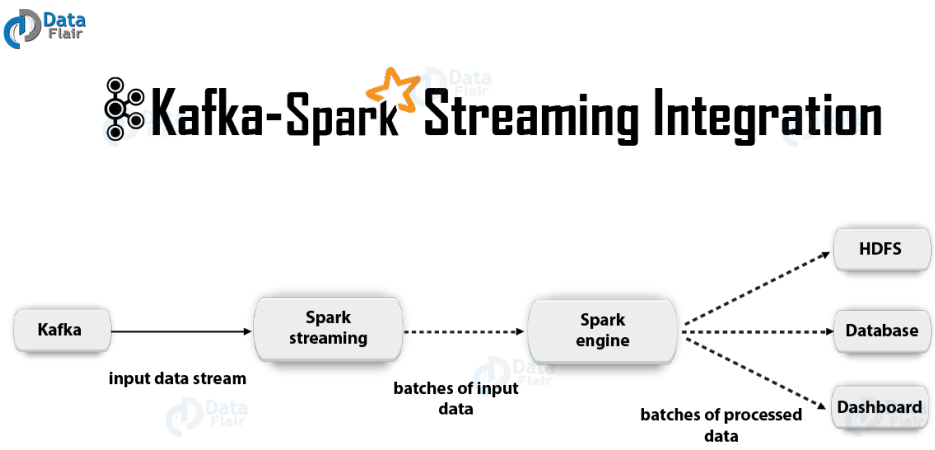


Figura 8.2: Integración de Kafka, Spark Streaming y Hadoop.

8.1.2 Infraestructura

Para el despliegue, el Centro de Investigación en Tecnologías de la Información y las Comunicaciones (CITIC) [58], nos proporciona 10 máquinas virtuales con las características resumidas en la tabla 8.1. Con el objetivo de aclarar qué papel toma cada una de ellas, a continuación, relacionamos las máquinas con el software que contienen y sus funciones principales:

- master: *Master Node* (Spark), *Master Node* (Hadoop).
- monitor: *Zabbix server*.

- worker1, worker2, worker3, worker4, worker5, worker6, worker7 y worker8: *Worker Node* (Spark), *Slave Node* (Hadoop), *Broker* (Kafka).

	Value
OS	Ubuntu 18.04 (Bionic Beaver)
Disk space	250 GB
RAM	16 GB
CPU(s)	4
Architecture	x86_64
Procesor Model	Intel(R) Xeon(R) Gold 6146 CPU @ 3.20GHz

Tabla 8.1: Especificaciones de las máquinas virtuales levantadas para el despliegue.

A fin de explicar mejor los roles de las máquinas, se puede observar el diagrama de la arquitectura Spark en la figura A.6, el de Hadoop en la figura A.7, y el de Kafka en la figura A.8. Zabbix se comenta en la sección 8.2.1.

8.2 Monitorización del sistema desplegado

8.2.1 Mecanismos de monitorización

Lo primero que tenemos que valorar es cuál de las opciones que pone a nuestra disposición Apache Spark para acceder a las estadísticas de rendimiento de nuestras aplicaciones es más adecuada. Existen varias alternativas [59]:

- Interfaz web sobre aplicación en ejecución: cuando ejecutamos una aplicación con Spark, que inicializa un *SparkContext*, se lanza también por defecto una interfaz web en el puerto 4040 del nodo controlador, en la que tenemos disponible información útil sobre la aplicación.

- Interfaz web sobre aplicación terminada: es posible usar la interfaz web también sobre el servidor de historias de Spark, que contiene los *logs* de las aplicaciones finalizadas. Para poder emplear este método, tenemos primero que activar los *logs* y configurar el servidor.

- API REST: también podemos obtener las métricas en formato JSON. De esta manera podemos crear herramientas de visualización y monitorización propias. Funciona tanto para aplicaciones corriendo como sobre el servidor de historias. Este método nos da acceso tanto a mediciones de las tareas para evaluación del rendimiento y carga de trabajo, como a mediciones de los ejecutores para describir el rendimiento de los mismos.

- Métricas: Spark cuenta con un sistema configurable de métricas que permite a los usuarios reportarlas a una variedad de *sinks*, que incluyen HTTP, JMX y CSV. Este sistema se puede configurar mediante un archivo que Spark busca en una ruta en concreto. Las métricas de Spark disponibles están agrupadas por instancias correspondientes a los componentes de Spark. Cada instancia puede reportar sus métricas a un conjunto de *sinks* que nosotros decidamos.

Aunque en las interfaces web de Spark se muestra información útil, no nos proporcionan toda la flexibilidad para visualizar los datos que nos gustaría. Por ello, aunque las consultemos de manera complementaria, no las consideramos el método principal de monitorización de nuestra aplicación. Ese papel lo lleva a cabo Zabbix [60].

Zabbix es una solución de monitorización de código abierto, gratis y de calidad empresarial. Puede supervisar parámetros de red y máquinas. Para analizar la información que obtiene y almacena, nos permite usar potentes herramientas de visualización. Soporta tanto el paradigma de *polling*, como el de *trapping*, y todos sus reportes y estadísticas, así como parámetros de configuración, están disponibles a través de una interfaz web. Esto permite que podamos comprobar el estado de nuestras máquinas desde cualquier localización.

Si bien tanto la API REST como el sistema de métricas nos permiten exportar la información relevante sobre nuestras aplicaciones, en este caso nos decantamos por el último. La razón principal es que mientras que la integración de Zabbix con el sistema de métricas de Apache Spark, a través de JMX, está perfectamente documentada [61] [62], no ocurre lo mismo con la API REST.

La tecnología JMX [63] provee todo lo necesario para construir soluciones de gestión y monitorización de aplicaciones distribuidas, basadas en web, modulares y dinámicas. Para activar el soporte de monitorización por JMX de Zabbix, hay que descargar e instalar el paquete con el demonio *'Zabbix Java gateway'*. En el lado de Apache Spark con nuestra aplicación, no tenemos que instalar nada, pero sí hay que indicar la activación del soporte para la monitorización remota JMX, de manera que la aplicación escuche las conexiones entrantes en el puerto correspondiente.

En la figura 8.3 podemos observar una vista general del panel de control o *dashboard* de Zabbix.

8.2.2 Resultados del sistema

Para evaluar el funcionamiento de la aplicación desplegada, se prueban dos configuraciones:

- Configuración 1: Spark se encarga tanto de la ingesta de datos como del procesamiento del *stream*. Los resultados de la predicción se escriben en un HDFS.

- Configuración 2: Kafka se encarga de la ingesta de datos, Spark del procesamiento del *stream*, y los resultados de la predicción se escriben en un HDFS.

Ambas configuraciones son sometidas a la ejecución de la aplicación con uno, cuatro, y ocho nodos trabajadores, de manera que, a partir de las métricas, se pueda comparar el comportamiento entre ejecuciones. Usamos las métricas *processingRate-total* e *inputRate-total*.

Los resultados que obtenemos son similares con las dos configuraciones. La aplicación se distribuye entre los nodos trabajadores, y el modelo es capaz de predecir a partir de los datos entrantes en *streaming*. Al contrario de lo que pensábamos en un principio, Kafka, tal y como lo incluimos, no supone una diferencia significativa con respecto a la configuración básica. También observamos que las métricas escogidas, *processingRate-total* e *inputRate-total*, no reflejan como se esperaba la fluctuación en el número de nodos. No se incluyen gráficas que ilustren los valores de estas métricas porque muestran un comportamiento similar independientemente de la configuración o cantidad de nodos que usemos.

Figura 8.3: Aspecto del panel de control de Zabbix una vez configurado.

Capítulo 9

Conclusiones y líneas futuras

A nivel personal, este proyecto ha supuesto una experiencia increíblemente enriquecedora. Poder llevar a cabo un trabajo de investigación sobre un tema tan actual e importante, es todo un privilegio. Además, la formación y experiencia adquiridas en el camino, complementan a la perfección lo aprendido tanto en el itinerario de Sistemas de Información, como en el de Tecnologías de la Información.

La motivación del proyecto fue comprobar que las técnicas de ML tienen la capacidad de detectar amenazas en las redes. La razón por la cual este tema ha sido un objeto de estudio de creciente interés en los últimos años, es la situación actual de las redes y la estimación de su evolución.

El primer paso para abordar esta tarea consistió en comprender cómo aplicar técnicas de ML a nuestras redes. También fue necesario averiguar de qué tipo de sistemas clásicos de protección partíamos, y cuáles eran las deficiencias que habían propiciado un cambio de paradigma.

Una vez clarificado el contexto, el foco de atención se desplazó hacia los datos. Los algoritmos de ML son capaces de extraer información de conjuntos de datos, pero su éxito en esa tarea está totalmente supeditado a la presentación de los mismos. Para saber cómo representarlos manera adecuada, antes tenemos que entenderlos en profundidad.

Con los datos entendidos, y teniendo en cuenta las peculiaridades de las técnicas elegidas, llegó el turno de seleccionar la información que íbamos a usar para modelar con cada uno de los algoritmos. Tanto la preparación y selección de los datos, como el modelado y su posterior hiperparametrización, son tareas profundamente dependientes de las características de las técnicas que empleamos. En consecuencia, para procurar los mejores resultados, tuvimos que adaptarnos a las necesidades de cada algoritmo.

Para la evaluación elegimos mecanismos de prueba adecuados para modelos de clasificación binaria, y establecimos cuál era la métrica que más importancia tenía para nosotros. En base a esto elegimos el mejor modelo de entre los que habíamos generado.

Hasta este punto, las actividades llevadas a cabo, relacionadas principalmente con el análisis e integración de datos, requirieron competencias específicas del itinerario de Sistemas de Información. Sin embargo, para las tareas de la última fase del ciclo de vida CRISP-DM, las competencias pasaron a estar mucho más orientadas al itinerario de Tecnologías de la Información.

En esta última fase se propuso, configuró y probó una infraestructura de despliegue distribuido para nuestro modelo. La evaluación se realizó sobre un *cluster* de diez máquinas, con el objetivo de obtener mediciones lo más realistas posible. Si bien los resultados no se alinearon con nuestras hipótesis iniciales, nos serán de utilidad para orientar nuestra investigación futura.

Consideramos que los objetivos del proyecto fueron alcanzados en su totalidad. Sin embargo, en este tipo de trabajos, siempre hay algo más que se puede hacer. En las líneas de investigación futura a continuación citadas, mencionamos algunas de las maneras en las que se podría continuar el trabajo realizado:

- Hiperparametrización que abarque un rango más amplio de búsqueda para DNN.
- Prueba de los algoritmos que dieron mejores resultados en otros conjuntos de datos.
- Empleo de técnicas de transformación de variables diferentes y más sofisticadas.
- Empleo de técnicas de selección de variables.
- Investigación más profunda sobre las razones que provocan que las configuraciones de despliegue no funcionen según lo esperado.
- Exploración de otras configuraciones de despliegue.
- Orquestación del despliegue con herramientas de administración automatizadas y basadas en contenedores.
- Implementación con MLlib y PySpark de toda la pipeline de transformaciones necesarias para que datos crudos puedan ser procesados en *streaming*, en vez de asumir que vienen preprocesados.

Apéndices

Apéndice A

Material adicional

A.1 Repositorio

https://git.fic.udc.es/julio.jairo.estevez.pereira/TFG

A.2 Atributos del *dataset* UNB ISCX 2012 en detalle

	count	unique	top	freq
appName	1885519	107	HTTPWeb	665441
sourcePayloadAsBase64	890269	487359	R0VUIC8gSFRUUC8xLjENCkhvc3Q6IDE5Mi4xNjguNS4xMj...	43475
sourcePayloadAsUTF	805466	325956	GET / HTTP/1.1Host: 192.168.5.122GET / HTTP/1....	43475
destinationPayloadAsBase64	802850	616573	KiBPSyBTdGlsbCBoZXJlDQoqIE9LIFN0aWxsIGhlcmUNCg==	10262
destinationPayloadAsUTF	802795	600905	* OK Still here* OK Still here	10262
direction	1885519	4	L2R	1613733
sourceTCPFlagsDescription	1481746	23	F,S,P,A	1041510
destinationTCPFlagsDescription	1423052	26	F,S,P,A	1028552
source	1885519	2473	192.168.5.122	246018
protocolName	1885519	6	tcp_ip	1484731
destination	1885519	33583	198.164.30.2	213478
startDateTime	1885519	294160	2010-06-13T16:42:25	9742
stopDateTime	1885519	318258	2010-06-13T16:42:25	9742
Target	1885519	2	Normal	1816609

Figura A.1: Descripción de las variables continuas iniciales del *dataset* UNB ISCX 2012.

	count	unique	top	freq
appName	1885519	107	HTTPWeb	665441
sourcePayloadAsBase64	890269	487359	R0VUIC8gSFRUUC8xLjENCkhvc3Q6IDE5Mi4xNjguNS4xMj...	43475
sourcePayloadAsUTF	805466	325956	GET / HTTP/1.1Host: 192.168.5.122GET / HTTP/1....	43475
destinationPayloadAsBase64	802850	616573	KiBPSyBTdGlsbCBoZXJlDQoqIE9LIFN0aWxsIGhlcmUNCg==	10262
destinationPayloadAsUTF	802795	600905	* OK Still here* OK Still here	10262
direction	1885519	4	L2R	1613733
sourceTCPFlagsDescription	1481746	23	F,S,P,A	1041510
destinationTCPFlagsDescription	1423052	26	F,S,P,A	1028552
source	1885519	2473	192.168.5.122	246018
protocolName	1885519	6	tcp_ip	1484731
destination	1885519	33583	198.164.30.2	213478
startDateTime	1885519	294160	2010-06-13T16:42:25	9742
stopDateTime	1885519	318258	2010-06-13T16:42:25	9742
Target	1885519	2	Normal	1816609

Figura A.2: Descripción de las variables categóricas iniciales del *dataset* UNB ISCX 2012.

A.3 Atributos del conjunto base en detalle

	count	mean	std	min	25%	50%	75%	max
totalSourceBytes	1632979.0	470.164569	373.919453	0.0	152.000000	3.900000e+02	561.000000	4466.0
totalDestinationBytes	1632979.0	4269.148446	8523.544178	0.0	258.000000	8.800000e+02	3337.000000	51094.0
totalSourcePackets	1632979.0	5.828261	4.675084	0.0	2.000000	5.000000e+00	7.000000	28.0
totalDestinationPackets	1632979.0	6.154499	6.806689	0.0	1.000000	5.000000e+00	7.000000	46.0
sourcePort	1632979.0	15542.988155	21058.285733	0.0	2367.000000	3.881000e+03	23082.000000	65535.0
destinationPort	1632979.0	2320.199774	9569.776427	0.0	80.000000	8.000000e+01	80.000000	65535.0
ratioSourceBytes	1632979.0	0.333937	0.277887	0.0	0.128602	2.742475e-01	0.410681	1.0
ratioDestinationBytes	1632979.0	0.666063	0.277887	0.0	0.589319	7.257525e-01	0.871398	1.0
xhr	1632979.0	0.015068	0.679717	-1.0	-0.707107	1.224647e-16	0.707107	1.0
yhr	1632979.0	-0.284110	0.676047	-1.0	-0.866025	-5.000000e-01	0.258819	1.0
sourceBitrate	1632979.0	2870.724194	3175.004223	0.0	824.000000	1.776000e+03	3920.000000	71456.0
destBitrate	1632979.0	19925.644885	48411.981538	0.0	1200.000000	5.536000e+03	16554.666667	802912.0
sourcePacketsRate	1632979.0	4.392664	4.662862	0.0	1.200000	2.500000e+00	6.000000	52.0
destPacketsRate	1632979.0	37.655446	50.373890	0.0	8.000000	2.000000e+01	48.000000	704.0
sourcePacketSize	1632979.0	84.928951	27.200069	0.0	73.800000	7.883333e+01	89.600000	1518.0
destPacketSize	1632979.0	3033.546065	2932.817133	0.0	979.200000	1.880000e+03	4208.000000	33176.0

Figura A.3: Descripción de las variables continuas finales empleadas para entrenar los modelos.

	count	unique	top	freq
SP	1632979	2	True	1036036
DP	1632979	2	True	1025618
SF	1632979	2	True	1032944
DF	1632979	2	True	1028476
SS	1632979	2	True	1061495
DS	1632979	2	True	1021208
SR	1632979	2	False	1598290
DR	1632979	2	False	1485039
SA	1632979	2	True	1189683
DA	1632979	2	True	1154232
udp_ip	1632979	2	False	1243288
tcp_ip	1632979	2	True	1235401
icmp_ip	1632979	2	False	1625165
S192.168.1	1632979	2	False	1333800
D192.168.1	1632979	2	False	1616704
S192.168.2	1632979	2	False	1038045
D192.168.2	1632979	2	False	1595010
S192.168.3	1632979	2	False	1447229
D192.168.3	1632979	2	False	1623062
S192.168.4	1632979	2	False	1349722
D192.168.4	1632979	2	False	1624803
S192.168.5	1632979	2	False	1380706
D192.168.5	1632979	2	False	1504631
S192.168.6	1632979	1	False	1632979
D192.168.6	1632979	1	False	1632979
Sothers	1632979	2	False	1632968
Dothers	1632979	2	False	1630378
Target	1632979	2	False	1601660

Figura A.4: Descripción de las variables categóricas finales empleadas para entrenar los modelos.

A.4 Importancia de los atributos en el modelo final de RF

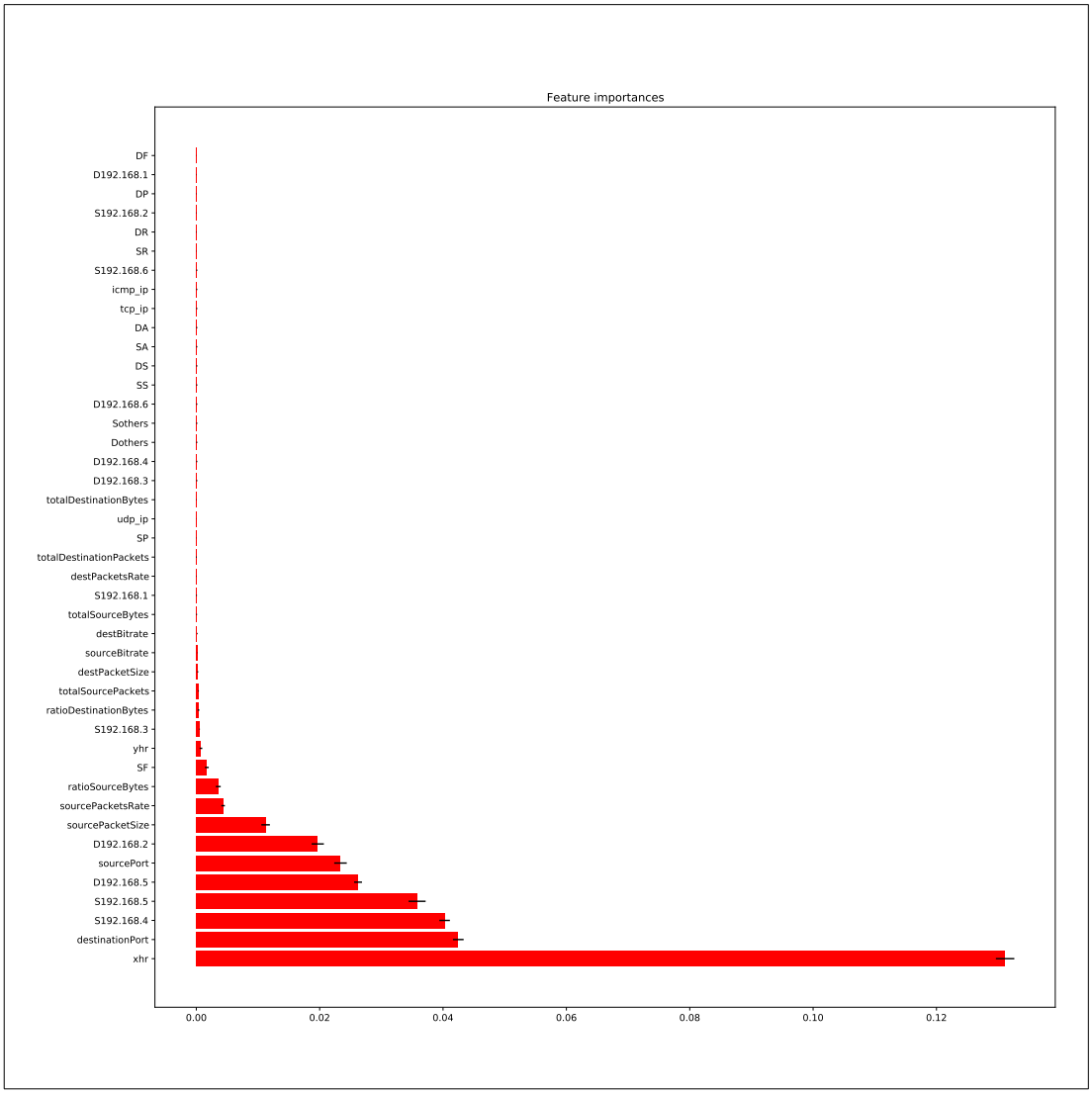


Figura A.5: Importancia que le da el modelo final de RF a cada atributo.

A.5 Parámetros finales de DNN

	Value
Batch size	60
Epochs	150
Optimization algorithm	Adam
Weight Initialization	RandomNormal
Activation Function	Softsign
Dropout rate (DR)	0.2
Weight constraint (DR)	4
Hidden Layers	3
Neurons per layer	160

Tabla A.1: Parámetros finales de DNN.

A.6 Arquitectura Spark

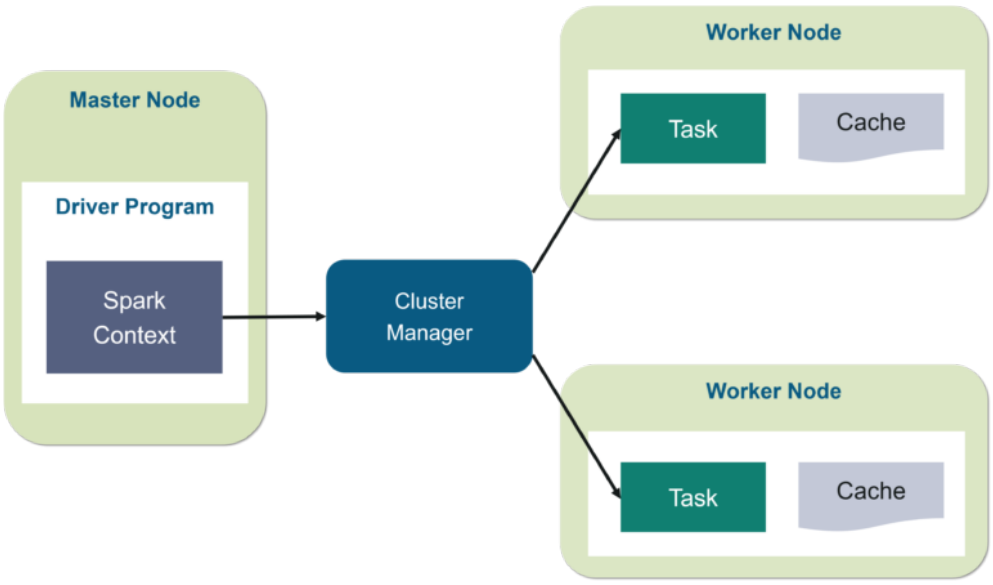


Figura A.6: Arquitectura de Spark [1].

A.7 Arquitectura Hadoop

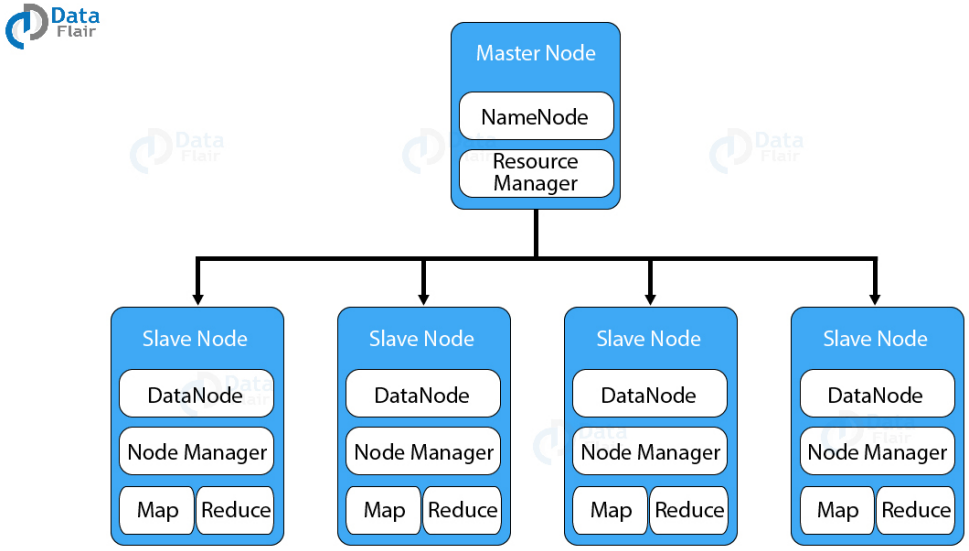


Figura A.7: Arquitectura de Hadoop [2].

A.8 Arquitectura Kafka

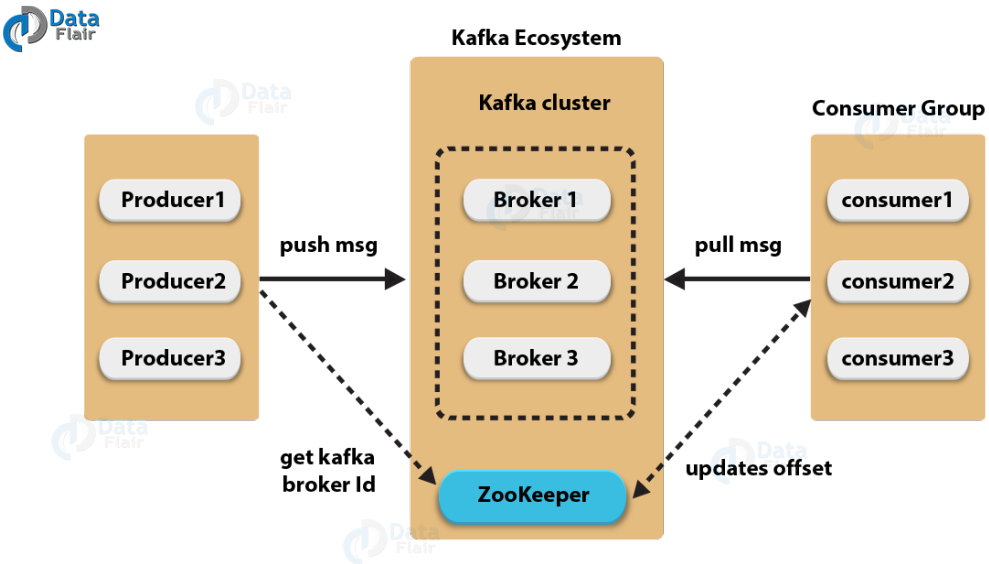


Figura A.8: Arquitectura de Kafka [3].

Lista de acrónimos

AD *Anomaly-based Detection.*

ANN *Artificial Neural Network.*

CRISP-DM *Cross Industry Standard Process for Data Mining.*

DDoS *Distributed Denial of Service.*

DL *Deep Learning.*

DNN *Deep Neural Network.*

DoS *Denial of Service.*

DPI *Deep Packet Inspection.*

EIF *Extended Isolation Forest.*

HD *Hybrid-most Detection.*

HDFS *Hadoop Distributed File System*

HIDS *Host-based IDS.*

ID *Intrusion Detection.*

IDS *Intrusion Detection System.*

IF *Isolation Forest.*

KDD *Knowledge Discovery in Databases.*

M2M *Machine to Machine.*

MIDS *Mixed IDS.*

ML *Machine Learning.*

NAT *Network Traffic Analysis.*

NB *Naïve Bayes.*

NBA *Network Behaviour Analysis.*

NIDS *Network based IDS.*

OHE *One-Hot-Encoding.*

OOB *Out Of Bag.*

R2L *Remote to Local*

R2U *Remote to User.*

ReLU *Rectified Linear Unit.*

RF *Random Forest.*

SD *Signature-based Detection.*

SEMMA *Sample, Explore, Modify, Model, and Assess.*

SPA *Stateful Protocol Analysis.*

U2R *User to Root.*

WIDS *Wireless-based IDS.*

Bibliografía

[1] N. Vaidya, "Apache spark architecture – spark cluster architecture explained." [Online]. Available: https://www.edureka.co/blog/spark-architecture/#:~:text=Apache%20Spark%20is%20an%20open,disk%20when%20compared%20to%20Hadoop.&text=Working%20of%20Spark%20Architecture

[2] D. Team, "Hadoop architecture in detail – hdfs, yarn & mapreduce." [Online]. Available: https://data-flair.training/blogs/hadoop-architecture/

[3] ——, "Kafka architecture and its fundamental concepts." [Online]. Available: https://data-flair.training/blogs/kafka-architecture/

[4] R. T. S. Project, "Internet live stats." [En línea]. Disponible en: https://www.internetlivestats.com/one-second/

[5] E. Koeze and N. Popper, "The virus changed the way we internet," *The New York Times*, 2020.

[6] ITU-R, "Report itu-r m.2370-0 (07/2015): Imt traffic estimates for the years 2020 to 2030," ITU, Tech. Rep., 2020.

[7] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19 – 31, 2016. [En línea]. Disponible en: http://www.sciencedirect.com/science/article/pii/S1084804515002891

[8] S. M. Othman, F. M. Ba-Alwi, N. T. Alsohybe, and A. Y. Al-Hashida, "Intrusion detection model using machine learning algorithm on big data environment," *Journal of Big Data*, vol. 5, no. 1, p. 34, 2018.

[9] Gartner, "Gartner glossary: Big data." [En línea]. Disponible en: https://www.gartner.com/en/information-technology/glossary/big-data#:~:

text=Big%20data%20is%20high%2Dvolume,decision%20making%2C%20and%20process%20automation.

[10] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An overview of ip flow-based intrusion detection," *IEEE communications surveys & tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[11] Gartner, "Gartner glossary: Distributed computing." [En línea]. Disponible en: https://www.gartner.com/en/information-technology/glossary/distributed-computing

[12] A. S. Tanenbaum and M. Van Steen, *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[13] V. K. Singh, "Key characteristics of distributed systems." [En línea]. Disponible en: https://medium.com/system-design-blog/key-characteristics-of-distributed-systems-781c4d92cce3

[14] D. Sun, Y. Zhang, and Y. Inoguchi, "Dynamic task flow scheduling for heterogeneous distributed computing: Algorithm and strategy," *IEICE Transactions on Information and Systems*, vol. E90D, 03 2007.

[15] S. Shahrivari, "Beyond batch processing: Towards real-time and streaming big data," *Computers*, vol. 3, pp. 1–2.7–8, 03 2014.

[16] A. Spark, "Spark streaming programming guide: Overview." [En línea]. Disponible en: https://spark.apache.org/docs/latest/streaming-programming-guide.html

[17] F. Foroughi and P. Luksch, "Data science methodology for cybersecurity projects," *CoRR*, vol. abs/1803.04219, 2018. [En línea]. Disponible en: http://arxiv.org/abs/1803.04219

[18] D. Rodríguez, M. F. Pollo Cattaneo, P. V. Britos, and R. García Martínez, "Estimación empírica de carga de trabajo en proyectos de explotación de información," in *XVI Congreso Argentino de Ciencias de la Computación*, 2010, pp. 664–668.

[19] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[20] G. Fernandez, "Deep learning approaches for network intrusion detection," Ph.D. dissertation, The University of Texas at San Antonio, 2019.

[21] P. Angelo and A. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Computing Surveys*, vol. 51, pp. 1–30, 05 2018.

[22] S. Yıldırım, "Naive bayes classifier — explained." [En línea]. Disponible en: https://towardsdatascience.com/naive-bayes-classifier-explained-50f9723571ed

[23] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[24] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[25] A. C. Müller, S. Guido *et al.*, *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.

[26] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[27] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[28] G. Golomb, "Is nta just another kind of ids?" [En línea]. Disponible en: https://awakesecurity.com/news/is-nta-just-another-kind-of-ids/

[29] Gartner, "Market guide for network traffic analysis." [En línea]. Disponible en: https://www.gartner.com/en/documents/3902353

[30] G. Piatetsky, "Python eats away at r: Top software for analytics, data science, machine learning in 2018: Trends and analysis." [En línea]. Disponible en: https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html

[31] M. de Empleo y Seguridad Social, "Boe-a-2018-3156." [En línea]. Disponible en: https://www.boe.es/eli/es/res/2018/02/22/(3)

[32] IANA, "Service name and transport protocol port number registry." [En línea]. Disponible en: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

[33] Wikipedia, "0.0.0.0." [En línea]. Disponible en: https://en.wikipedia.org/wiki/0.0.0.0

[34] Keycdn, "Tcp flags." [En línea]. Disponible en: https://www.keycdn.com/support/tcp-flags

[35] M. Fukushima and S. Goto, "Analysis of tcp flags in congested network," *IEICE Transactions on Information and Systems*, vol. E83-D, pp. 996–1002, 05 2000.

[36] Keycdn, "What should be the allowed percentage of missing values?" [En línea]. Disponible en: https://discuss.analyticsvidhya.com/t/what-should-be-the-allowed-percentage-of-missing-values/2456

[37] E. Lewinson, "Outlier detection with isolation forest." [En línea]. Disponible en: https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e

[38] G. A. Susto, A. Beghi, and S. McLoone, "Anomaly detection through on-line isolation forest: An application to plasma etching," *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 89–94, 2017.

[39] E. Lewinson, "Outlier detection with extended isolation forest." [En línea]. Disponible en: https://towardsdatascience.com/outlier-detection-with-extended-isolation-forest-1e248a3fe97b

[40] Groud, "Implement extended isolation forest #16517." [En línea]. Disponible en: https://github.com/scikit-learn/scikit-learn/issues/16517

[41] M. DelSole, "What is one hot encoding and how to do it." [En línea]. Disponible en: https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179

[42] D. Kaleko, "Feature engineering - handling cyclical features." [En línea]. Disponible en: http://blog.davidkaleko.com/feature-engineering-cyclical-features.html

[43] C. Dossman, "Feature engineering - handling cyclical features." [En línea]. Disponible en: https://medium.com/ai%C2%B3-theory-practice-business/top-6-errors-novice-machine-learning-engineers-make-e82273d394db

[44] B. Roy, "All about feature scaling." [En línea]. Disponible en: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

[45] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[46] Imblearn, "imblearn.under_sampling.randomundersampler." [En línea]. Disponible en: https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html

[47] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[48] UNB, "Intrusion detection evaluation dataset (cic-ids2017)." [En línea]. Disponible en: https://www.unb.ca/cic/datasets/ids-2017.html

[49] Scikit-learn, "sklearn.model_selection.gridsearchcv." [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[50] J. Brownlee, "How to grid search hyperparameters for deep learning models in python with keras." [En línea]. Disponible en: https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

[51] Google, "Machine learning glossary." [En línea]. Disponible en: https://developers.google.com/machine-learning/glossary

[52] J. Brownlee, "How to develop and evaluate naive classifier strategies using probability." [En línea]. Disponible en: https://machinelearningmastery.com/how-to-develop-and-evaluate-naive-classifier-strategies-using-probability/

[53] K. Asel Mendis, "Naive bayes: A baseline model for machine learning classification performance." [En línea]. Disponible en: https://www.kdnuggets.com/2019/04/naive-bayes-baseline-model-machine-learning-classification-performance.html#:~:text=Naive%20Bayes%20is%20a%20supervised,inspired%20by%20the%20Bayes%20theorem.&text=Naive%20Bayes%20is%20a%20classification,class%20to%20make%20a%20prediction.

[54] A. Spark, "Mllib." [En línea]. Disponible en: https://spark.apache.org/mllib/

[55] ——, "Hadoop." [En línea]. Disponible en: https://hadoop.apache.org/

[56] ——, "Apache kafka." [En línea]. Disponible en: https://kafka.apache.org/

[57] D. Team, "Apache kafka + spark streaming integration." [En línea]. Disponible en: https://data-flair.training/blogs/kafka-spark-streaming-integration/

[58] CITIC, "Centro de investigación en tecnologías de la información y las comunicaciones." [En línea]. Disponible en: https://www.citic-research.org/

[59] A. Spark, "Monitoring and instrumentation." [En línea]. Disponible en: https://spark.apache.org/docs/latest/monitoring.html

[60] Zabbix, "What is zabbix." [En línea]. Disponible en: https://www.zabbix.com/documentation/2.0/manual/introduction/about

[61] ——, "Jmx monitoring." [En línea]. Disponible en: https://www.zabbix.com/documentation/3.4/manual/config/items/itemtypes/jmx_monitoring

[62] ——, "Java gateway." [En línea]. Disponible en: https://www.zabbix.com/documentation/3.4/manual/concepts/java

[63] Oracle, "Welcome to the jmx technology home page." [En línea]. Disponible en: https://www.oracle.com/java/technologies/javase/javamanagement.html