



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

Detección de anomalías de red mediante técnicas de machine learning

Estudiante: Fernando Javier Villar Freire
Dirección: Francisco Javier Nóvoa Manuel
Dirección: Diego Fernández Iglesias

A Coruña, novembro de 2019.

*Por ser mi faro y mi refugio durante nuestra travesía te dedico todo este esfuerzo a ti, Vane.
Para que en el futuro sigas siendo mi luz.*

Agradecimientos

Quiero agradecer de todo corazón a toda mi familia el apoyo y el esfuerzo que han realizado para que yo haya podido finalizar el grado y conseguir entregar este trabajo fin de grado. Sin ellos nada de esto hubiese sido posible. Espero poder compensar con creces todo lo que me fue dado.

Resumen

El panorama actual [1], con un aumento del tráfico debido a la proliferación del internet de las cosas, al auge de la filosofía del siempre conectado, el aumento de los servicios multimedia, el *cloud*, la videoconferencia..., ha provocado la proliferación de ataques y de nuevas amenazas. Debido a ello, surge la necesidad de nuevos modelos de defensa rápidos y adaptables.

Existen diferentes estrategias a la hora de afrontar el estudio del tráfico de red, pero el uso del enfoque de agregación, tanto en la captura como en el análisis, viene a solucionar en parte los problemas de las ingentes cantidades de tráfico que soportan a día de hoy las organizaciones. La agregación permite agrupar el tráfico, en un momento dado, en una secuencia de paquetes que comparten unos valores determinados, consiguiendo de esta forma reducir el volumen de datos a tratar sin perder la información relevante de los mismos.

A pesar de la reducción en el volumen de información sigue haciéndose necesario el uso de técnicas de *Big Data* para analizarla. El auge actual de las técnicas de *machine learning* junto con el crecimiento, disponibilidad y sencillez de uso de la computación distribuida nos proporcionan las herramientas necesarias para acometer las tareas centradas en el análisis de tráfico agregado.

El objetivo principal de este proyecto, a través de una prueba de concepto, es comparar y comprender diferentes técnicas de *machine learning* aplicadas a la detección en tiempo real de comportamientos anómalos en el tráfico de una red. De manera transversal incidiremos en aspectos como la captura de flujos o la implementación y despliegue de una solución distribuida como herramientas para dotar a nuestro modelo de las características esperadas de rendimiento y escalabilidad.

Abstract

In the current scenario [1] with an increase of traffic (due to the proliferation of the internet of things, the rise of the philosophy of always connected and the increase in multimedia services, cloud, videoconferencing and its associated traffic) along with the proliferation of attacks and new threats, there is a need for fast and adaptable defense models.

There are different approaches when dealing with the study of network traffic, but the use of the aggregation approach, both in the capture as in the analysis, comes to partially solve the problems of the huge amounts of traffic that support day-to-day traffic of the organizations. The aggregation allows grouping the traffic, at any given time, in a sequence of packets that share certain values, getting this way reduce the volume of data to be processed without losing the relevant information.

Despite the reduction in the volume of information, achieved through the use of flows, the use of Big Data techniques to analyze it is still necessary. The current boom in machine learning techniques, hereinafter ML, together with the growth, availability and simplicity of use of distributed computing, provide us with the necessary tools to undertake the tasks focused on the analysis of aggregate traffic.

The main objective of this project is to compare and understand different ML techniques applied to the real-time detection of anomalous behaviors in a network's traffic. In a transversal way we will focus on aspects such as the capture of flows or the implementation and deployment of a distributed solution as tools to provide our model with the expected characteristics of performance and scalability..

Palabras clave:

- Machine learning
- Clasificación
- Ingeniería de características
- Regresión logística
- Random forest
- SVM
- Anomalia
- IDS
- Computación distribuida
- Flujos

Keywords:

- Machine learning
- Clasification
- Feature engineering
- Logistic regression
- Random forest
- SVM
- Anomaly
- IDS
- Distributed computing
- Flows

Índice general

1	Introducción	1
1.1	¿Pero de qué forma se relacionan <i>machine learning</i> y la ciberseguridad?	2
1.2	¿Qué es una anomalía y cómo identificarla?	4
1.3	La problemática del <i>Big Data</i>	5
1.3.1	Flujos	6
1.3.2	Computación distribuida	6
1.4	Objetivos	9
1.5	Organización del proyecto	9
2	Metodología	11
2.1	CRISP-DM	12
3	Entender el negocio	15
3.1	<i>Machine learning</i>	16
3.1.1	¿Qué es lo que quiero hacer?	16
3.1.2	¿Qué información tengo para conseguir mi objetivo?	18
3.2	Algoritmos	18
3.2.1	Regresión logística	19
3.2.2	<i>Random Forest</i>	20
3.2.3	<i>Support Vector Machine: SVM</i>	22
3.3	Netflow e IPfix	23
3.3.1	IPfix (<i>IP Flow Information Export</i>)	25
3.4	IDS y NAT	25
3.4.1	La anatomía de un NTA	26
3.5	¿Por qué DataBricks?	27
3.6	¿Por qué Python/PySpark?	29
3.7	¿Por qué Weka?	29
3.8	Planificación	29

3.8.1	<i>Scrum</i>	29
4	Entender los datos	33
4.1	Descripción	33
4.2	Recolección	34
4.3	Exploración	35
5	Preparar los datos	41
5.1	Limpieza	41
5.1.1	Diagrama de caja	42
5.1.2	<i>Isolation forest</i>	43
5.2	Transformación	45
5.2.1	Creación	45
5.2.2	Escalado	46
5.2.3	Discretización de valores	48
5.3	Selección	48
5.3.1	Regresión logística	50
5.3.2	<i>Random Forest</i>	52
5.3.3	SVM	53
6	Modelar	55
6.1	Asunciones sobre los modelos	55
6.2	Validación	56
6.2.1	Métricas de rendimiento	56
6.2.2	Validación cruzada	58
6.3	Hiperparametrización	59
6.3.1	Regresión logística	59
6.3.2	<i>Random Forest</i>	62
6.3.3	SVM	64
7	Evaluar	69
8	Desplegar	73
8.1	Rendimiento	75
9	Conclusiones y líneas futuras	79
A	Material adicional	83
A.1	<i>Sprints</i> del proyecto	83
A.2	Descripción de los atributos iniciales del <i>dataset</i>	85

ÍNDICE GENERAL

A.3	Repositorio GIT	96
A.4	Selección de atributos	96
A.4.1	Regresión logística	96
A.4.2	<i>Random forest</i>	100
A.4.3	SVM	104
A.5	Hiperparametrización	108
A.5.1	Random forest	108
A.5.2	SVM	111
A.6	Modelos finales	114
A.6.1	Regresión logística	114
A.6.2	<i>Random forest</i>	117
A.6.3	SVM	120
	Lista de acrónimos	125
	Bibliografía	127

Índice de figuras

1.1	Porcentaje de implantación del <i>cloud computing</i> frente al de centros de datos tradicionales (<i>datacenters</i>)	1
1.2	Ideas claves de la arquitectura CARTA	2
1.3	Diagrama PPDR	3
1.4	Identificar una anomalía	4
1.5	Previsión de crecimiento del tráfico IP	5
1.6	Arquitectura y configuración de un sistema de captura de flujos	6
1.7	Diagrama básico de un sistema distribuido	7
1.8	Esquema de funcionamiento de <i>MapReduce</i>	8
1.9	Consecuencias en la monitorización debido al intervalo de agregación	8
1.10	Funcionamiento de un sistema distribuido basado en flujos	9
2.1	Diagrama KDD	11
2.2	Diagrama CRISP-DM	12
2.3	Metodología CRISP-DM	13
3.1	Fase CRISP-DM: Entender el negocio.	15
3.2	Plantilla de elección de algoritmos de ML de Microsoft Azure	19
3.3	Función sigmoidea.	20
3.4	Proceso de regresión logística para clasificación binaria.	20
3.5	Construcción de un árbol de decisión.	21
3.6	Margen máximo respecto al hiperplano en SVM.	23
3.7	Arquitectura y configuración de un sistema de captura de flujos	24
3.8	Componentes de Apache Spark.	27
3.9	Funcionamiento de Apache Spark.	28
3.10	Marco <i>Scrum</i> técnico	31
4.1	Fase CRISP-DM: Entender los datos.	33

4.2	Escenario del <i>dataset</i> UNB ISCX IDS 2012	34
4.3	Información general del <i>dataset</i>	35
4.4	Distribución de la clase objetivo	36
4.5	Valores presentes en el atributo <i>destinationTCPFlagsDescription</i>	37
4.6	Valores con <i>flag</i> <i>Nan</i> en tráfico TCP en el atributo <i>destinationTCPFlagsDescription</i>	38
4.7	Valores con <i>flag</i> <i>N/A</i> en tráfico TCP en el atributo <i>destinationTCPFlagsDescription</i>	38
4.8	Comando <i>nmap</i> para mostrar el top 20 de puertos escaneados de los protocolos TCP y UDP	39
4.9	Servicios de nuestro <i>dataset</i> coincidentes con el top <i>nmap</i>	39
5.1	Fase CRISP-DM: Preparar los datos.	41
5.2	Diagrama de cajas y datos anómalos(<i>outliers</i>)	42
5.3	Diagrama de caja del atributo <i>timelength</i> agrupado por <i>HTTPWeb</i>	43
5.4	Diferencia entre identificar muestras anómalas o identificar muestras normales	43
5.5	Distribución de los valores <i>timelength</i> en el tráfico normal	44
5.6	Resultado del uso de isolation Forest en el tráfico normal <i>HTTPWeb</i>	44
5.7	Cambios tras escalar en el atributo <i>avgDestinationPacketSize</i>	47
5.8	Cambios tras realizar un remuestreo 1:1	50
5.9	Muestras de número de ataques agrupados por <i>appName</i> y <i>Tag</i>	51
5.10	Diagrama de correlación de los principales atributos numéricos	51
5.11	Detalle de la salida del método de selección de atributos para el <i>dataset</i> de <i>Random Forest</i>	53
6.1	Fase CRISP-DM: Modelar.	55
6.2	Precisión (<i>Accuracy</i>) vs Exactitud (<i>Precision</i>)	57
6.3	Validación cruzada y estratificación	59
6.4	Distribución de valores del atributo <i>typeAttack</i>	60
6.5	Flujo de modelado de Regresión Logística	60
6.6	Resultado final del modelo basado en regresión logística	61
6.7	Coeficientes de la función de regresión	62
6.8	Hiperparámetros candidatos y finales para <i>Random Forest</i>	62
6.9	Flujo de modelado de <i>Random Forest</i>	63
6.10	Resultado final del modelo basado en <i>Random Forest</i>	64
6.11	Comparativa de <i>kernels</i> respecto a la exactitud	65
6.12	Comparativa de <i>kernels</i> respecto al tiempo	65
6.13	Hiperparámetros candidatos y finales para SVM	66

ÍNDICE DE FIGURAS

6.14	Flujo de modelado de SVM	66
6.15	Resultado de la validación cruzada del modelo basado en SVM	67
7.1	Fase CRISP-DM: Evaluar	69
7.2	Comparativa final entre regresión logística y <i>Random Forest</i>	70
7.3	Comparativa sobre la validación del <i>dataset</i> completo entre regresión logística y <i>Random Forest</i>	71
8.1	Fase CRISP-DM: Desplegar	73
8.2	<i>Clusters</i> en Databrick	74
8.3	Exactitud del modelo <i>Random Forest</i> en Databricks	75
8.4	<i>Cluster</i> funcionando con dos trabajadores	76
8.5	<i>Cluster</i> funcionando con cuatro trabajadores	76
8.6	Problemas de balanceo en <i>clusters</i> de pocos trabajadores	77
A.1	Resumen visual del dataset resultante para el método de regresión logística . .	100
A.2	Resumen visual del dataset resultante para el método de <i>Random Forest</i>	104
A.3	Resumen visual del dataset resultante para el método de SVM	108

Índice de tablas

6.1	Resumen del estado del <i>dataset</i>	56
6.2	Matriz de confusión	56
A.1	Tareas del <i>sprint 0</i>	83
A.2	Tareas del <i>sprint 1</i>	83
A.3	Tareas del <i>sprint 2</i>	84
A.4	Tareas del <i>sprint 3</i>	84
A.5	Tareas del <i>sprint 4</i>	84
A.6	Tareas del <i>sprint 5</i>	85

Capítulo 1

Introducción

La transformación digital a la que se está sometiendo la sociedad ha provocado que especialmente gobiernos y empresas hayan abrazado las bondades ofrecidas por los paradigmas de la nube, la movilidad y el siempre conectado. Entre otras ventajas destacan una mayor respuesta al cambio y un ahorro de costes.

La implantación del *cloud computing* [2] (figura 1.1) ha llevado a reemplazar el actual modelo centrado en la seguridad perimetral de un *datacenter* a un modelo orientado a la seguridad usuario-aplicación [3]. No quiere decir que el concepto de perímetro haya desaparecido, sino que se ha ampliado hasta las fronteras de los dispositivos y las aplicaciones que corren sobre ellos. El perímetro se puede encontrar en cualquier parte. Mayor perímetro supone mayores riesgos y mayores complejidades.

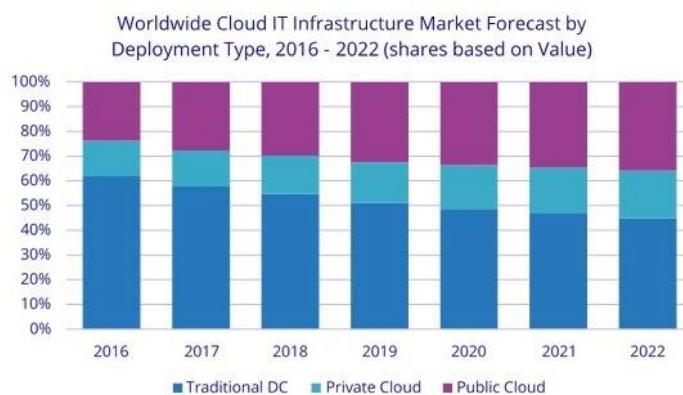


Figura 1.1: Porcentaje de implantación del *cloud computing* frente al de centros de datos tradicionales (*datacenters*)

Como respuesta a este cambio Gartner, considerado uno de los mayores referentes y fuente de consulta en el ecosistema TI, presenta en el año 2014 *Adaptive Security Architecture* como un plan para actualizar a las organizaciones en materia de seguridad y adaptarlas a los tiempos

1.1. ¿Pero de qué forma se relacionan machine learning y la ciberseguridad?

venideros. La idea ha evolucionado a la actual CARTA (*Continuous adaptive risk and trust assessment*), representada en la figura 1.2, una filosofía pensada para ser implementada desde el departamento de *DevOps* hasta los socios externos de la organización [4].

La implantación de CARTA lleva a las organizaciones a hacer especial énfasis en la analítica de datos y el *machine learning*, en adelante ML, asociado a ella. Aumentar la automatización y disminuir el tiempo de respuesta, todo ello sin que conlleve un aumento de recursos humanos, son unas de las ventajas asociadas a su uso.

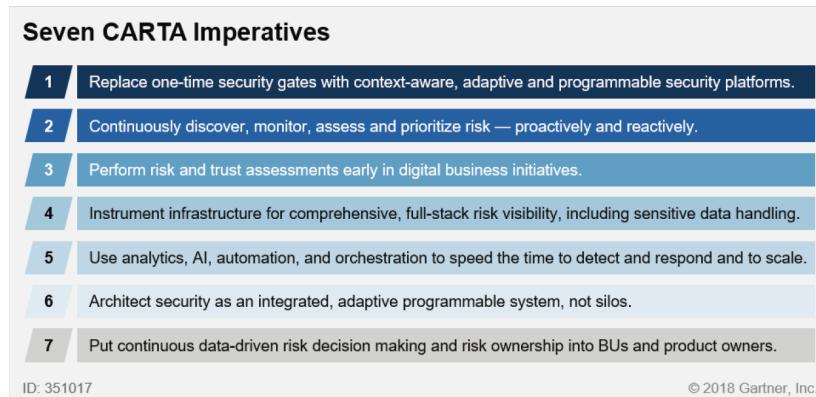


Figura 1.2: Ideas claves de la arquitectura CARTA

1.1 ¿Pero de qué forma se relacionan *machine learning* y la ciberseguridad?

A diferencia con otras áreas como pueden ser el reconocimiento de imágenes y el procesado del lenguaje natural en donde ML es la herramienta indiscutible, en ciberseguridad las sinergias se originan debido a que una gran parte de las tareas asociadas a la seguridad pueden ser implementadas a través de métodos de ML. Podemos observar esto tomando como punto de partida el modelo PPDR de Gartner (figura 1.3) y listando las tareas asociadas a sus cinco puntos:

- Predicción
- Prevención
- Detección
- Respuesta
- Vigilancia

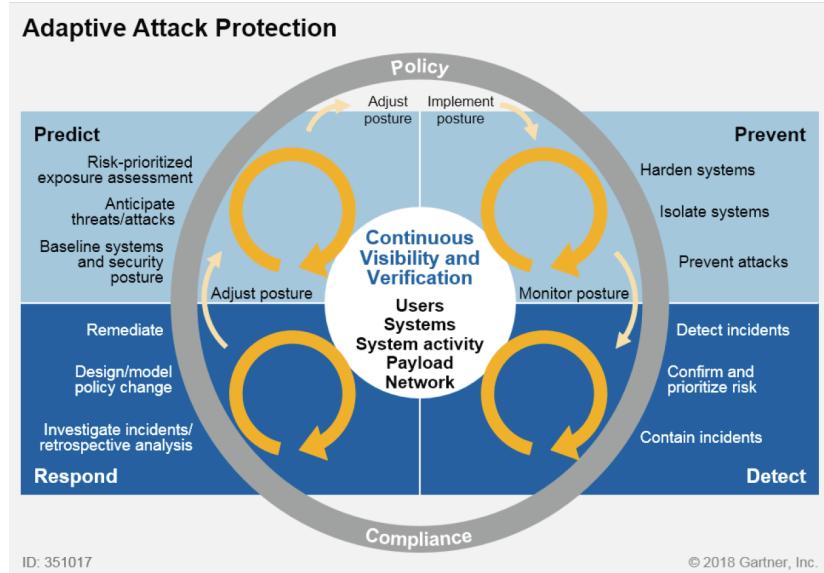


Figura 1.3: Diagrama PPDR

ML puede abordar estas tareas a través de los diferentes algoritmos y métodos de regresión, predicción y clasificación de los que dispone. Algunos ejemplos podrían ser:

- Técnicas de regresión para predecir los parámetros de los paquetes y comparar estos con los que el sistema considera normales.
- Técnicas de clasificación para identificar tipos de ataques y tráfico anómalo.
- Técnicas de *clustering* orientadas al análisis forense.

En este proyecto abordaremos aspectos referidos a la protección de la red, concretamente las tareas necesarias para identificar el tráfico anómalo que discurre por ella. Aproximaciones clásicas como los sistemas de detección de intrusiones [5] (IDS) muestran la preocupación de las organizaciones acerca de la seguridad. Dentro de los sistemas IDS existen dos grandes aproximaciones: los basados en firmas y los basados en detección de anomalías. Los primeros se basan en identificar a través de patrones (firmas) los posibles ataques. Su efectividad está acotada a ataques ya conocidos. La segunda familia se centra en estudiar el tráfico y detectar desviaciones anormales del mismo, lo cual le permite detectar ataques todavía desconocidos. Una vuelta de tuerca a este concepto es la denominación NTA (*Network Traffic Analysis*), creada y definida por Gartner, que da un paso más allá y centra su funcionamiento en las metodologías ML como medio para detectar e investigar amenazas de seguridad y comportamientos anómalos o maliciosos dentro de una red.

En febrero de 2019, Gartner publica el primer informe sobre NTA [6] en el que se incluye, además de una retrospectiva sobre la tecnología, una lista de los 17 suministradores de servicio

más representativos. En dicho informe se hace hincapié de nuevo en que las herramientas que se quieran englobar en esta nueva categoría han de hacer uso de técnicas de ML o similares y no de modelos basados en firmas.

1.2 ¿Qué es una anomalía y cómo identificarla?

Podemos definir anomalía [7, cap. 1] como todo aquello que se escapa de lo esperado. Es una definición vaga, que por sí misma no nos permite identificarla, pero nos permite adivinar su principal característica: las anomalías no se definen por sus propias características sino por contraposición sobre lo que sí es normal. Antes de identificar una anomalía es necesario saber qué es normal (figura 1.4).

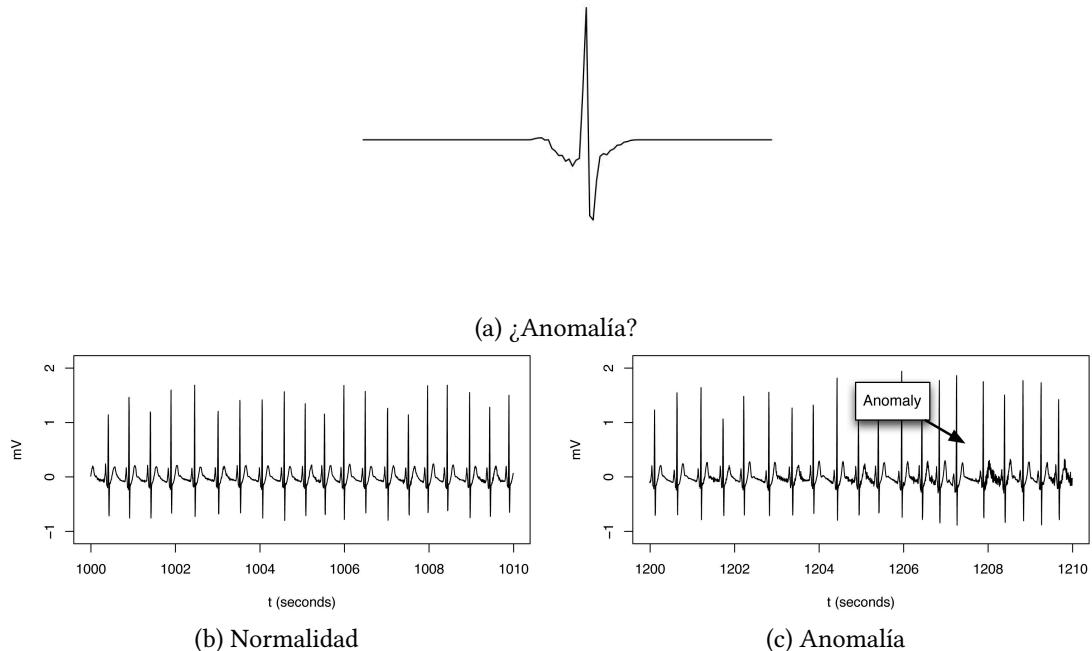


Figura 1.4: Identificar una anomalía

En nuestro contexto el término hace referencia a comportamientos inesperados de acuerdo al funcionamiento normal de la red. No debemos asumir que tráfico anómalo es igual a tráfico malicioso. Existen diferentes escenarios, como pueden ser los ocasionados por el funcionamiento inadecuado del hardware, que producen anomalías en el comportamiento de la red sin que por ello tengan un carácter malicioso.

En la definición inicial se describe el mecanismo principal detrás de la detección de anomalías. No se puede saber previamente cómo es una anomalía, pero sí se puede construir un sistema que caracterice el patrón esperado para identificar lo que es diferente del patrón.

Consiste en encontrar lo que no se sabe que se busca. La detección de anomalías determina el comportamiento esperado y considera anomalías todo aquello que se desvíe significativamente de este.

Las técnicas de ML ligadas a la clasificación o el *clustering*, permiten generalizar comportamientos de un conjunto de datos por medio de la obtención de patrones. Esto las convierte en un mecanismo idóneo para identificar casos que no se conocían previamente, un funcionamiento en concordancia con la definición dada de detección de anomalías.

1.3 La problemática del *Big Data*

La cantidad de tráfico generada en la red debido al uso de nuevos paradigmas (*cloud*, internet de las cosas...) ha crecido enormemente a lo largo de estos últimos años y la tendencia (figura 1.5) prevé que siga en aumento.

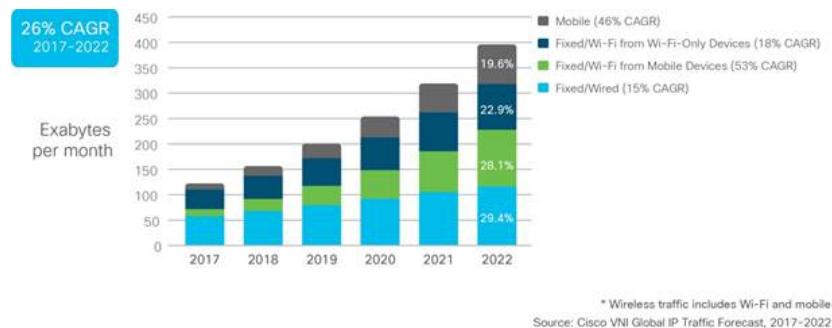


Figura 1.5: Previsión de crecimiento del tráfico IP

A tenor de la definición de *Big Data* [8] como todo conjunto de datos demasiado complejo y/o voluminoso para que su almacenamiento y procesamiento no sea sencillo, podemos concluir que el volumen de datos referentes al contexto de nuestro proyecto, análisis del tráfico de red en tiempo real, se adecúa a tal definición. Por lo tanto, nos encontramos ante la disyuntiva de poseer una cantidad ingente de datos que nos permite análisis precisos y pormenorizados pero que a su vez nos imposibilita trabajar con ellos de una forma rápida y eficiente.

La monitorización es el primer paso para la realización de las tareas de análisis. En un enfoque tradicional, centrado en la captura de paquetes, presenta dos grandes problemas que son las dos caras de una misma moneda: por una parte, el volumen de datos a almacenar y procesar, y por otra el ancho de banda a gestionar. En el último punto es importante destacar que la monitorización y análisis en el contexto del proyecto se requiere en tiempo real o lo más cercano a él. Con redes que pueden superar los 100 Gbps la captura de paquetes y su análisis suponen un desafío para las infraestructuras de la red en términos de almacenamiento y capacidad de cómputo.

Existen varias aproximaciones para su solución, pero el uso de la agregación y la computación distribuida son las principales.

1.3.1 Flujos

La solución a la reducción de los requerimientos de almacenamiento en nuestro contexto se centra en la técnica de agregación del tráfico a partir de la creación de flujos. Esta agregación permite agrupar el tráfico, en un momento dado, en una secuencia de paquetes que comparten unos valores determinados. A esta agrupación le llamamos flujo [9]. El uso de flujos permite trabajar sólo con la información que consideremos relevante y descartar toda la demás si lo consideramos conveniente. El uso de la captura de flujos no es excluyente de la captura de paquetes.

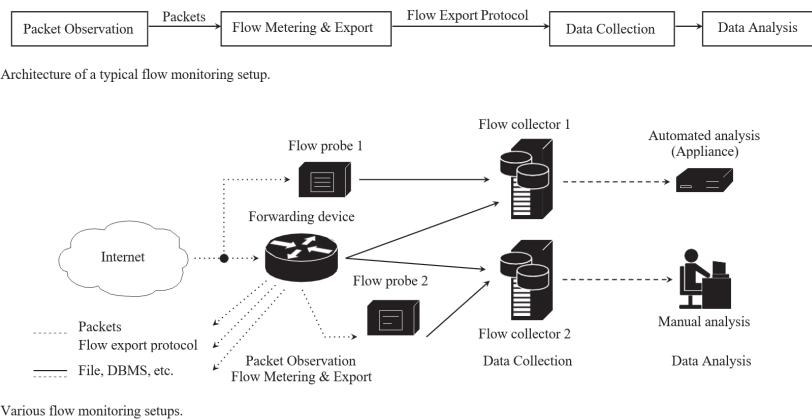


Figura 1.6: Arquitectura y configuración de un sistema de captura de flujos

El uso de flujos para la captura de tráfico conlleva diferentes ventajas, pero en este punto podemos destacar dos principalmente:

- Bajo coste de implantación: esto es debido a que gran parte del hardware de las infraestructuras de red, cerca de un 70% [10], soportan de forma nativa diferentes protocolos que permiten la agregación de tráfico.
- Reducción drástica del volumen de datos: son posibles ratios de 1/2000 respecto al volumen original tras la agregación de los paquetes capturados [11].

1.3.2 Computación distribuida

La computación distribuida es el campo de las ciencias de la computación dedicadas al estudio de los sistemas distribuidos [12, cap. 1]. Un sistema distribuido está compuesto por un

conjunto de máquinas, independientes entre sí, interconectadas mediante una red de interconexión a los que un determinado software convierte en un sistema de mayores prestaciones orientado a alcanzar un objetivo común. En la computación distribuida cada nodo de computación suele poseer su propia memoria (a no ser en el caso de la existencia de memoria distribuida), los nodos se encuentran sobre una red y su comunicación y coordinación se establece a través de mensajes enviados por ella (figura 1.7).

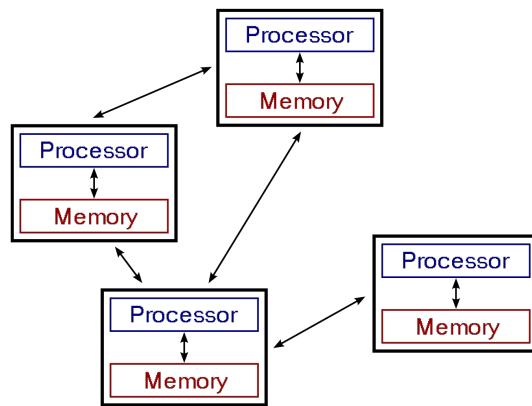


Figura 1.7: Diagrama básico de un sistema distribuido

Existen diferentes arquitecturas que permiten implementar un sistema distribuido, pero en el contexto del *Big Data* la más utilizada es la basada en *batch*, lotes de procesamiento de datos. Esta aproximación permite ejecutar una aplicación no paralelizable en un sistema distribuido de manera sencilla. En concreto, el paradigma *MapReduce* (figura 1.8) permite desarrollar algoritmos distribuidos, capaces de escalar, solo definiendo dos funciones: *map* y *reduce* [12, cap. 3].

- *Map*: procesa los datos de entrada y genera pares clave:valor.
- *Reduce*: procesa los valores asociados a una determinada clave.

Sistemas distribuidos basados en flujos

A pesar de que el uso de flujos mejora el costo computacional debido a la reducción de elementos inherente a la técnica de agregación, su uso conlleva una problemática asociada a que no es una técnica 100% en tiempo real.

La aproximación habitual al análisis de flujos es a través de *batchs* donde los datos se procesan a intervalos, generalmente de 5 minutos [13]. Este enfoque introduce retrasos y variaciones en el análisis de los datos que disminuye la eficiencia en las tareas de detección de anomalías o la identificación de amenazas. Cuando hablamos de variaciones nos referimos

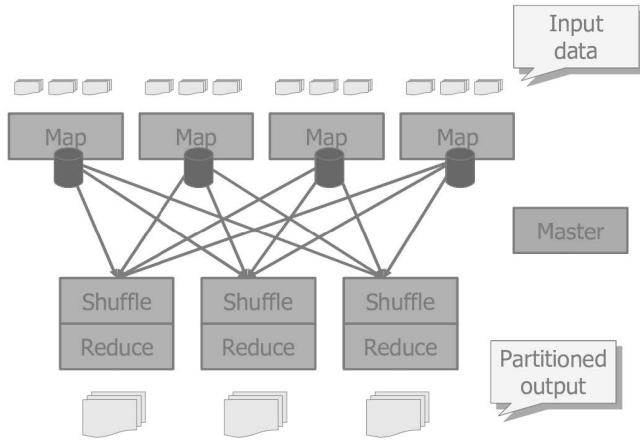


Figura 1.8: Esquema de funcionamiento de *MapReduce*

a que el uso de flujos lleva a que ciertas características del tráfico de red analizado, como pueden ser los picos, queden ocultas (figura 1.9). Al ser estos picos posibles anomalías en el comportamiento del tráfico, es una desventaja muy a tener en cuenta en el ámbito de nuestro proyecto.

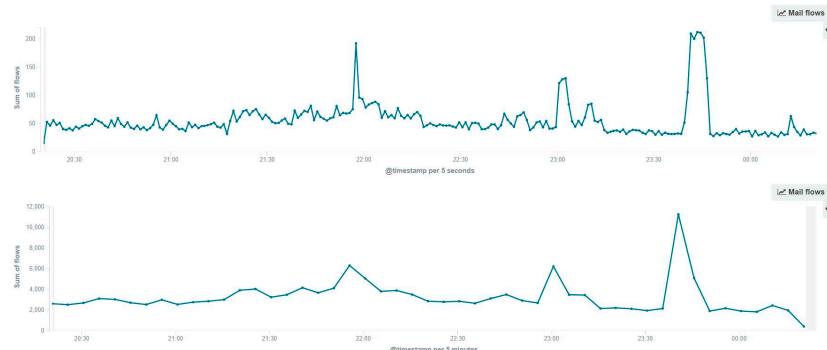


Figura 1.9: Consecuencias en la monitorización debido al intervalo de agregación

La solución es disminuir el tiempo de agregación y procesar los flujos en tiempo real. Para conseguir procesar los datos en tiempo real y con la rapidez suficiente, acorde al ancho de banda, es necesario añadir al paradigma de computación distribuida el concepto de flujo (*stream*).

El uso de flujos en la computación distribuida es diferente a su aproximación tradicional basada en *batchs*. En esta ocasión los datos no se almacenan de forma persistente en un disco o en memoria, sino que existen como una secuencia de mensajes. El sistema se ha de encargar de capturarlos, procesarlos y generar el resultado final del cálculo.

La dificultad de las diferentes implementaciones de sistemas basados en flujos se centra

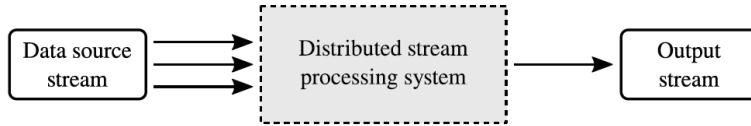


Figura 1.10: Funcionamiento de un sistema distribuido basado en flujos

en la obtención de resultados en verdadero tiempo real y conseguir ser escalables respecto al aumento del volumen de datos y la complejidad de los cálculos a realizar sobre ellos.

1.4 Objetivos

A continuación, listamos los objetivos a alcanzar por el proyecto con el fin de realizar una prueba de concepto que implemente la funcionalidad básica de un NTA (sección 3.4): la detección de anomalías en el tráfico monitorizado.

- Estudiar la arquitectura de agregación de flujos y sus principales protocolos: proporcionaremos una visión global del estado actual de la arquitectura y sus principales protocolos.
- Seleccionar, implementar y comparar diferentes algoritmos de ML: realizaremos la selección de los algoritmos para nuestra tarea de clasificación como paso previo a las tareas de ingeniería de características sobre el *dataset* de estudio. Para finalizar realizaremos un estudio de los resultados obtenidos con cada uno de los algoritmos seleccionados.
- Configurar y desplegar un sistema distribuido que albergue los modelos de ML finales: expondremos los conceptos claves a la hora de abordar la configuración y puesta en marcha de la arquitectura distribuida que sirva de sustento al despliegue de nuestros modelos de ML.
- Evaluar el rendimiento de los modelos de ML: ejecutaremos pruebas de rendimiento en nuestra plataforma distribuida por cada uno de los modelos finales de ML. Nos centraremos en el número de flujos que pueda procesar el modelo y la escalabilidad del mismo.

1.5 Organización del proyecto

- Cap. 2 - Metodología:
Aspectos metodológicos usados para la realización del proyecto.

- Cap. 3 - Entender el negocio:
 - Explicación detallada del conocimiento necesario para entender y llevar a cabo el proyecto.
 - Explicación y justificación detallada sobre las herramientas y las tecnologías usadas para la consecución del proyecto.
 - Exposición de las alternativas presentes que abordan la temática del proyecto
 - Planificación del proyecto detallando cada una de las fases de ejecución.
- Cap. 4 - Entender los datos:
Explicación centrada en las tareas referidas a la recolección inicial, descripción y explotación de los datos.
- Cap. 5 - Preparar los datos:
Explicación centrada en las tareas referidas a la selección, limpieza y transformación de los datos.
- Cap. 6 - Modelar:
Explicación centrada en las tareas de selección de modelos de ML, detallando los procesos de hiperparametrización, entrenamiento y validación.
- Cap. 7 - Evaluar:
Explicación detallada de las pruebas necesarias para la correcta validación de las funcionalidades del proyecto
- Cap. 8 - Desplegar:
Explicación de la puesta en funcionamiento de los modelos sobre el sistema distribuido y análisis de las pruebas finales de rendimiento.
- Cap. 9 - Conclusiones y líneas futuras:
 - Objetivos alcanzados y lecciones aprendidas tras la conclusión del proyecto.
 - Funcionalidades futuras por implementar o desafíos por abordar.

Capítulo 2

Metodología

Existen diferentes metodologías a la hora de abordar un proyecto de minería de datos como puede ser un trabajo basado en ML. Todas ellas tienen como meta definir las fases necesarias para realizar de forma exitosa y eficaz los proyectos. Dos de los principales exponentes son KDD (*Knowledge Discovery in Database*) y CRISP-DM (*Cross-industry standard process for data mining*) ambos alumbrados a finales de los años 90 [14, 15]. Los dos describen procesos interactivos en los que algunas de sus fases están concebidas como puntos de reentrada en el proceso con el fin de conseguir un refinamiento final.

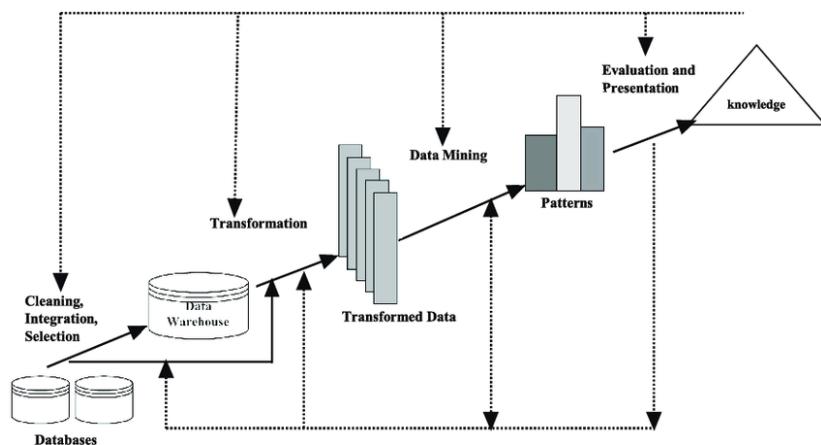


Figura 2.1: Diagrama KDD

A alto nivel, ambos modelos resultan similares, pero con una visión más cercana, podemos apreciar que mientras KDD se centra sobre todo en las tareas a realizar, CRISP-DM se enfoca más en el proceso. Otra diferencia es que CRISP-DM [16] presenta un marcado enfoque al proceso de negocio.

En este proyecto nos hemos decantado por CRISP-DM, ya que sus diferentes fases se acoplaban perfectamente a la estructura del TFG.

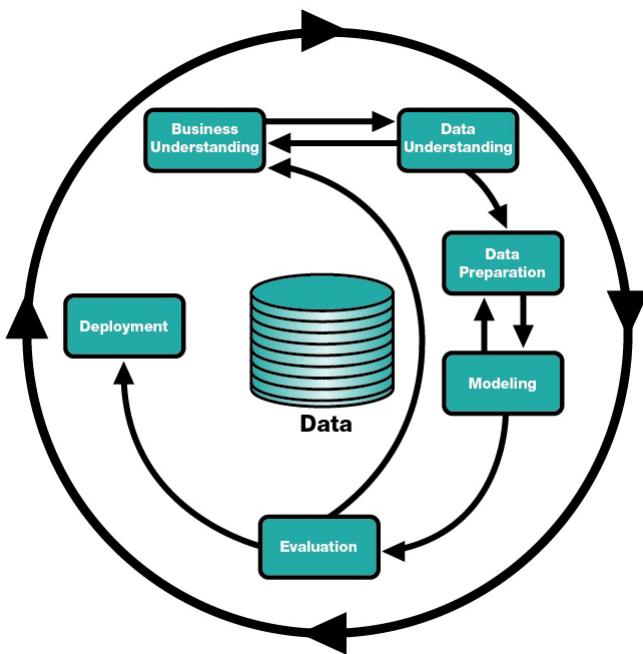


Figura 2.2: Diagrama CRISP-DM

2.1 CRISP-DM

Nacido en 1996 como una iniciativa del fondo europeo para dar soporte a los proyectos de minería de datos, presenta dos características que han posibilitado su gran acogida en el sector:

- No propietario.
 - No dependiente de tecnologías o herramientas.

La metodología CRISP-DM consta de seis fases diferenciadas formadas por múltiples tareas (figura 2.3). Los dos primeros niveles abarcan todos los procesos de minería de datos que están presentes en un proyecto de estas características. El tercer nivel hace referencia a aquellas tareas propias de una tipología específica de proyecto.

La descripción de las seis fases es la siguiente:

- Entender el negocio: centrada en determinar los objetivos del proyecto desde el punto de vista del negocio. Como resultado se obtiene un plan de proyecto (definición y requerimientos) y la determinación de los resultados esperados al fin del mismo.
 - Entender los datos: centrada en la recolección inicial, la descripción y exploración de los datos. Como resultado se tendrá una visión de la calidad de los datos y el conocimiento subyacente a ellos.

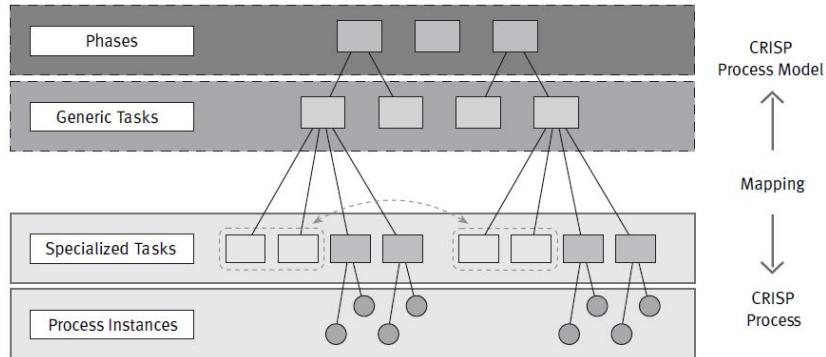


Figura 2.3: Metodología CRISP-DM

- Preparar los datos: es la fase que concentra la mayor carga de trabajo. ML se centra en extraer conocimiento de los datos, los datos son la clave y en torno a lo que gira todo el proyecto, por lo que el tiempo dedicado a estas tareas supone un beneficio claro en el resultado final. Las actividades principales dentro de esta fase son la selección, limpieza y transformación de los datos. Como resultado obtenemos un conjunto de datos susceptibles de ser modelados.
- Modelar: engloba las tareas dedicadas a creación del modelo: selección, hiperparametrización, entrenamiento y validación. La selección del algoritmo dependerá de la naturaleza del proyecto y los datos que poseemos para su consecución. Como resultado se obtiene un posible modelo que cumpla los requisitos del proyecto.
- Evaluar: encargada de evaluar si los resultados obtenidos son los esperados y están alineados con los objetivos de negocio (rendimiento, costes, tiempo). Aunque no es una característica exclusiva de esta fase, en ésta se valora de manera expresa la necesidad de repetir o rehacer algunos o todos los anteriores pasos con el fin de acercarse al objetivo esperado. Como resultado se obtiene un modelo listo para su despliegue.
- Desplegar: realiza el despliegue del modelo ya validado y las tareas propias de un proyecto en producción como son la monitorización y el mantenimiento. Como resultado se obtiene el informe final del proyecto.

Como comentamos con anterioridad, se trata de un proceso iterativo donde la secuencia de las fases no es estricta y puede trastocarse con el fin de refinar el resultado. De este modo las flechas de la figura 2.2 se deben tomar como las dependencias más destacadas y no como la indicación de un flujo inamovible.

Usaremos el modelo CRISP-DM como hilo conductor para presentar las funcionalidades implementadas y realizadas durante el proyecto. Debemos aclarar que, en la redacción de la memoria, con el fin de mantener un relato continuista y reducir su extensión, se ha omitido

el carácter iterativo del modelo CRISP-DM, donde los cambios realizados en el *dataset* se deberían probar con el fin de ir refinando el modelo final; es decir, no se han plasmado las diferentes iteraciones que han sido realizadas durante las fases del proyecto.

Capítulo 3

Entender el negocio

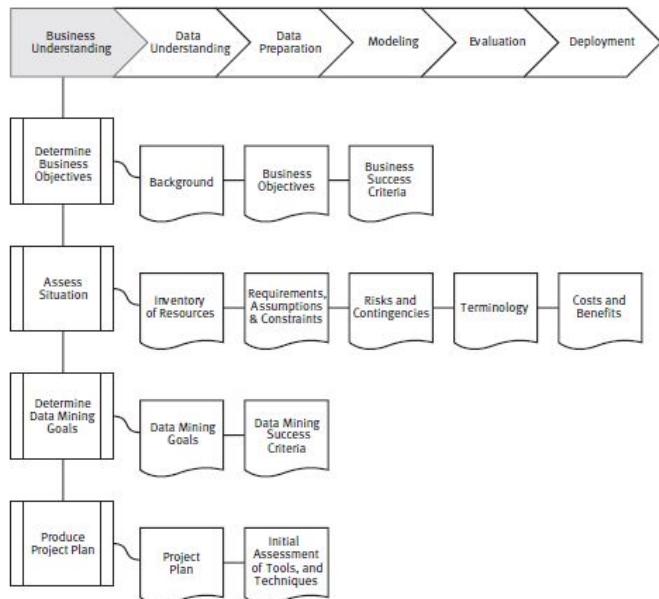


Figura 3.1: Fase CRISP-DM: Entender el negocio.

ENTENDER el negocio implica conocer los fundamentos teóricos y tecnológicos, así como las herramientas existentes para la consecución de los objetivos que nos hemos propuesto. Las tareas fundamentales de esta fase son:

- Determinar los objetivos de negocio: esta tarea la hemos abordado al definir los objetivos del proyecto expuestos en la introducción 1.4 de este documento.
- Evaluar la situación: tarea que nos marca el contexto de trabajo para la realización del proyecto. En las siguientes secciones 3.1, 3.2, 3.3, 3.4, 3.5, 3.6 y 3.7 profundizaremos en cada una de las opciones que hemos tenido en cuenta a la hora de abordar el TFG.

- Determinar los objetivos a conseguir: resumidos en el propio título de nuestro proyecto "Detección de anomalías de red".
- Desarrollar un plan de proyecto: completamente abordado en la sección 3.8 donde se especifica tanto la metodología del plan como la concreción del mismo.

3.1 Machine learning

Machine Learning [17] es una de las muchas aproximaciones que usa la Inteligencia Artificial (IA) a la hora de extraer conocimiento de la experiencia, no sólo como herramienta para conseguir los objetivos ligados a la IA sino como vehículo para reducir el tiempo y el esfuerzo empleado para su resolución. ML reconoce patrones a través de la utilización de ejemplos en lugar de abordar su implementación. La máquina aprende un modelo a partir de ejemplos y usa éste para resolver el problema. Un sistema que aprende continuamente, que es capaz de tomar decisiones basadas en los datos en lugar de los algoritmos y que cambia su comportamiento, es un sistema basado en ML.

Dado un problema la elección del algoritmo adecuado no es trivial. Para ello es necesario plantearse dos preguntas fundamentales: *¿qué es lo que quiero hacer?* y *¿qué información tengo para conseguir mi objetivo?* Ambas nos obligan a realizar un análisis exhaustivo del propio problema y de los datos, cantidad y calidad, que disponemos para su resolución.

3.1.1 ¿Qué es lo que quiero hacer?

Contestar a dicha pregunta nos proporcionará la tarea a acometer. La tarea resultante se podrá englobar en una de las tareas listadas a continuación o en una de las derivadas de ellas. Existen diferentes tareas asociadas al ML pero solo nos centraremos, debido a la naturaleza del proyecto, en aquellas que guardan una mayor relación con la ciberseguridad.

- Regresión: tarea que tiene como objetivo predecir un nuevo valor a partir de los valores anteriores. En el ámbito de la ciberseguridad es usada en la detección de fraudes. Características como la localización, el número de transacciones sospechosas, importes, etc., determinan la probabilidad de que una determinada transacción se corresponda a un fraude.

Algunos de los métodos más comunes de ML para su resolución son los siguientes: regresión lineal, regresión polinómica, árboles de decisión, *Random Forest* y *Support Vector Regression* (SVR).

- Clasificación: tarea que tiene como objetivo separar elementos en diferentes categorías. En el ámbito de la ciberseguridad su mejor ejemplo y posiblemente la primera

implementación real son los filtros de clasificación de correo no deseado. Es importante destacar que nos encontramos ante un método supervisado donde las categorías son conocidas de antemano.

Algunos de los métodos más comunes de ML para su resolución son los siguientes: regresión logística, *K-Nearest Neighbors* (K-NN), *Support Vector Machine* (SVM), *Naive-Bayes*, árboles de decisión y *Random Forest*.

- Agrupamiento o *clustering*: tarea similar a la clasificación en la que se desconocen las categorías objetivo y el agrupamiento se realiza por similitud. En esta ocasión nos encontramos con un método no supervisado. Su uso suele estar ligado a ser un elemento dentro del flujo de resolución más que a ser la tarea única de resolución de un problema. En el ámbito de la ciberseguridad su mayor uso se realiza durante el análisis forense por que durante la ejecución de dichas tareas, las razones, consecuencias y otros parámetros de los incidentes son desconocidos. Se necesita clasificar todas las actividades afectadas para encontrar anomalías.

Algunos de los métodos más comunes de ML para su resolución son los siguientes: *K-Nearest Neighbors* (K-NN), *K-means* y *Bayesian*.

- Reglas de asociación o sistemas de recomendación: tarea que tiene como objetivo recomendar algo en base a experiencias previas. En el ámbito de la ciberseguridad se suele usar como método de automatizar la respuesta a incidentes. El sistema aprende un tipo de respuesta para un incidente en particular y frente a nuevos casos, que no han de ser iguales, actúa en consecuencia (asignación de la gravedad de una vulnerabilidad a partir sólo de su descripción).

Algunos de los métodos más comunes de ML para su resolución son los siguientes: *Apriori*, *Euclat*, *FP-Growth*.

- Reducción de la dimensionalidad o generalización: tarea que tiene como objetivo buscar las características comunes y más relevantes dentro de una colección de datos. Su uso se hace necesario en tareas que presentan un número elevado de características y los métodos que las implementan tienen limitadas el número de ellas de las que pueden hacer uso. Reducción de la dimensionalidad permite eliminar aquellas características que no son relevantes. Al igual que *clustering*, reducción de la dimensionalidad suele ser una tarea más dentro de la resolución de un problema complejo. En el ámbito de la ciberseguridad su uso aparece ligado a las soluciones de reconocimiento facial.

Algunos de los métodos más comunes de ML para su resolución son los siguientes: *Principal Component Analysis* (PCA), *Singular-Value Decomposition* (SVD), *Independent Component Analysis* (ICA).

3.1.2 ¿Qué información tengo para conseguir mi objetivo?

El conocimiento extraído de los datos es la clave del funcionamiento de ML. La presencia o ausencia de etiquetas que definen previamente a cada uno de los datos del conjunto permite que la misma tarea de ML sea abordada desde varios enfoques.

- Aprendizaje supervisado: se parte de datos etiquetados que mediante algoritmos asignan la etiqueta de salida adecuada de entre las previamente definidas. Su mayor limitación es la necesidad inicial de tener un conjunto de datos correctamente etiquetado.

Este enfoque es especialmente útil para los problemas de clasificación y regresión, ya que en ellos diferenciamos la variable objetivo siendo categórico en el primer caso y numérico en el segundo.

- Aprendizaje no supervisado: el entrenamiento emplea datos sin etiquetar por lo cual solo es posible describir su estructura y mediante su descripción encontrar algún tipo de organización que permita un análisis más profundo. Las etiquetas de salida no pueden conocerse a priori.

Este enfoque está indicado para problemas de *clustering* y su mayor ventaja es el ahorro de tiempo y la posibilidad de solventar la falta de información.

- Aprendizaje por refuerzo: usaremos este enfoque cuando los algoritmos ML no encajen en ninguno de los anteriores. Mediante la retroalimentación el sistema aprende a base de ensayo-error fomentando el aprendizaje mediante refuerzos positivos y negativos.

No es aprendizaje supervisado, ya que no se basa en un conjunto de datos etiquetados, sino en la monitorización de la respuesta a las acciones tomadas. Tampoco es no supervisado, ya que el modelo previamente deberá conocer cuál es el refuerzo esperado.

3.2 Algoritmos

Elegir el algoritmo necesario para resolver el problema puede efectuarse mediante plantillas de ayuda.

Siguiendo la plantilla de la figura 3.2, escogemos para el problema que nos compete tres algoritmos de clasificación:

- Regresión logística, porque devuelve resultados fácilmente interpretables en un tiempo de entrenamiento reducido.
- *Random Forest*, por su gestión del ruido y su comportamiento con conjuntos de datos de gran tamaño.

- SVM, que es la mejor elección para conjuntos de datos con gran cantidad de atributos y resiste con eficacia la presencia de datos anómalos.

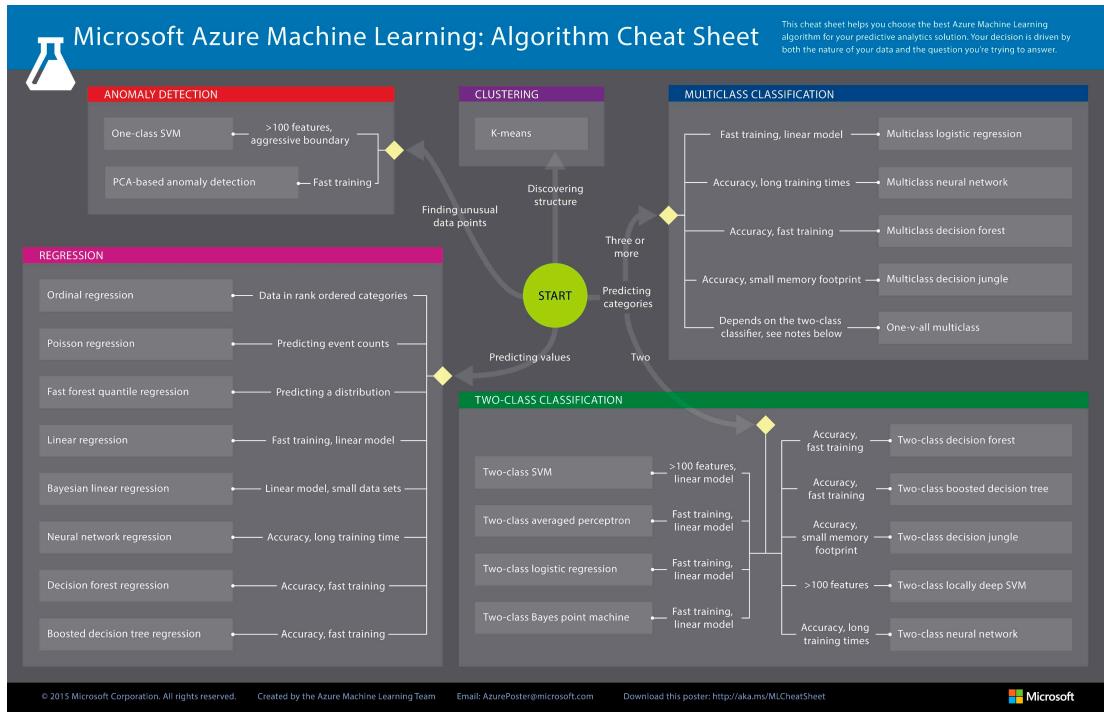


Figura 3.2: Plantilla de elección de algoritmos de ML de Microsoft Azure

3.2.1 Regresión logística

La regresión logística [18, cap.6] es un método de ML proveniente del ámbito de la estadística y ampliamente usado para resolver problemas de clasificación binaria. Es un tipo de análisis de regresión que permite predecir el resultado de una variable categórica binaria en función de las variables independientes o predictoras.

El método se divide en dos fases. En la primera por medio de una función lineal se predice la probabilidad condicionada en función de los valores de entrada, y en una segunda fase una función no lineal garantiza que los valores se encuentren en el intervalo [0,1]. Hay que tener en cuenta que la diferencia entre la regresión logística y lineal es que la regresión logística brinda un resultado discreto, pero la regresión lineal proporciona un resultado continuo. La función encargada de este paso es la función logística o sigmoidea, que es la que da nombre al método.

Pros:

- Las salidas del algoritmo tienen una interpretación probabilística.

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

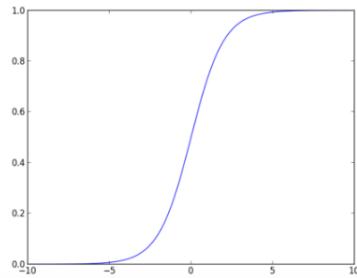


Figura 3.3: Función sigmoidea.

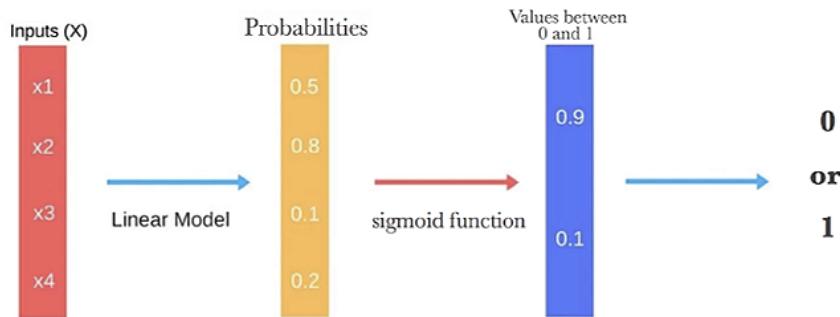


Figura 3.4: Proceso de regresión logística para clasificación binaria.

- Los modelos se pueden actualizar fácilmente con nuevos datos utilizando el descenso de gradiente estocástico.

Contras:

- La regresión logística tiende a tener un rendimiento inferior cuando hay límites de decisión múltiples o no lineales.
- No es lo suficientemente flexible como para capturar relaciones complejas.
- Extremadamente susceptible a la presencia de datos anómalos.

3.2.2 Random Forest

Random Forest [18, cap.8] constituye una mejora sobre los árboles de decisión. El algoritmo de árbol de decisión se emplea tanto para tareas de clasificación como para regresión. El árbol se construye dividiendo repetidamente el espacio de los datos en dos subconjuntos (figura 3.5), pudiendo manejar una combinación de entradas discretas y continuas.

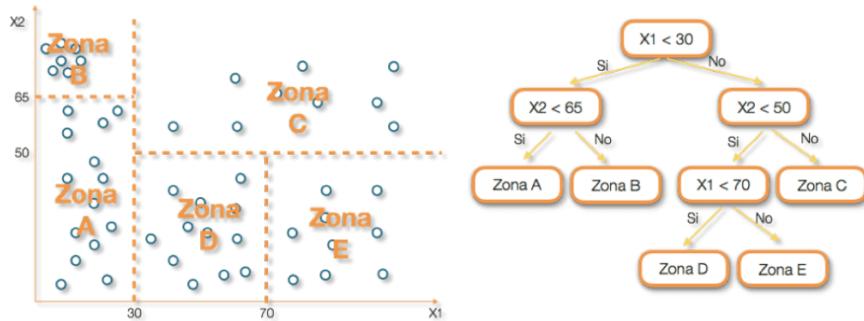


Figura 3.5: Construcción de un árbol de decisión.

Permite representar de manera visual y explícita la toma de decisiones y es capaz de manejar grandes conjuntos de datos y subsanar la existencia de valores atípicos o ausentes. Todo esto lo convierte en uno de los métodos más empleado en minería de datos.

Pros:

- Simple y rápido.
- Produce resultados interpretables.
- Permite manejar datos numéricos y categóricos.

Contras:

- Pueden dar lugar a árboles demasiado complejos o árboles inestables dado que pequeñas variaciones en los datos conforman árboles totalmente diferentes.

Random Forest es un algoritmo de aprendizaje supervisado que construye múltiples árboles de decisión y los fusiona para obtener una predicción más precisa y estable. Es un tipo de algoritmo que usa técnicas combinadas [19] (*bootstrap*, *bagging*...) para conseguir predicciones más exactas respecto a modelos individuales. Una desventaja del uso de *bagging* es que los árboles de decisión se construyen utilizando un algoritmo codicioso que selecciona el mejor punto de división en cada paso del proceso de construcción del árbol por lo cual los árboles resultantes son muy similares, lo que perjudica la solidez de las predicciones realizadas. *Random Forest* soluciona esta situación limitando de forma aleatoria la muestra de atributos para cada árbol creado. El trabajo con *Random Forest* evita los problemas de sobreajuste que presentan los árboles de decisión, ya que estos últimos no se podan, sino que se generan en profundidad por lo que plantean tantas similitudes estructurales que las predicciones están demasiado correlacionadas. *Random Forest*, al seleccionar de forma aleatoria el subconjunto de atributos de entrada, disminuye la similitud entre los árboles y, a su vez, las predicciones resultantes. De esta manera, aunque se incremente el tiempo necesario para su entrenamiento conseguimos minimizar en gran medida el problema del sobreajuste.

Pros:

- Simplicidad y versatilidad para tareas de clasificación y regresión.
- Reduce el sobreajuste.
- Permite manejar datos numéricos y categóricos.
- Permite trabajar con conjunto de datos incompletos y con valores anómalos.

Contras:

- La proliferación de árboles de decisión puede hacer que el algoritmo sea lento e ineficaz para las predicciones en tiempo real.
- Es difícil visualizar el modelo por lo cual se hace menos interpretable.
- Resulta costoso a nivel computacional.

3.2.3 *Support Vector Machine: SVM*

Support Vector Machine [20] es un algoritmo de aprendizaje automático que funciona buscando una línea que separe los datos en dos grupos. Esta línea se llama hiperplano y divide el espacio de la entrada separando los puntos de entrada por su clase. Esto se realiza por medio de un proceso de optimización que únicamente considera las instancias de datos de entrenamiento más cercanas al hiperplano. Solo estos puntos denominados vectores de soporte son relevantes para definir el hiperplano y construir el clasificador. SVM halla los coeficientes que den como resultado la separación óptima de las clases por el hiperplano partiendo de la idea de que todos nuestros puntos de entrada pueden ser completamente separados por esta línea. Este algoritmo hace clasificaciones usando este hiperplano y evalúa los nuevos valores de entrada sobre la ecuación del mismo calculando la posición del nuevo punto respecto a la linea de separación.

La distancia entre el hiperplano y los puntos de datos más cercanos se denomina margen. El hiperplano óptimo es aquel que maximiza la distancia de separación entre las clases. Esto se llama hiperplano de margen máximo. En la práctica, los datos reales son confusos y no se pueden separar perfectamente con un hiperplano dando lugar a clasificadores de margen suave que permite cierto grado de relajación en la separación de los datos. La desventaja es que algunos de los datos puedan sobrepasar la línea de separación por lo cual es necesario introducir un parámetro de ajuste llamado C que defina la magnitud de oscilación permitida. Cuanto mayor sea el valor de C mayores serán los puntos que sobrepasen el hiperplano.

El algoritmo SVM se implementa en la práctica usando *kernels*. Estos permiten modificar el comportamiento a la hora de definir el hiperplano. Los *kernels* pueden ser de varias clases:

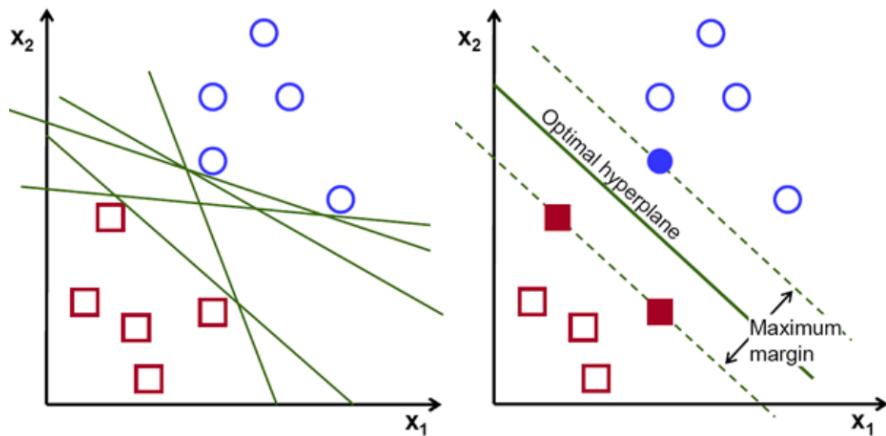


Figura 3.6: Margen máximo respecto al hiperplano en SVM.

las formas más básicas crearán hiperplanos a través de funciones lineales, pero en casos de datos complejos la separación no es posible por lo cual debemos usar *kernels* que trasformen el espacio de entrada en dimensiones superiores, como pueden ser los *kernels* polinómicos o radiales. De esta forma el hiperplano resultante se ajustará mejor a los datos de entrada.

Pros:

- Resistente al sobreajuste especialmente en conjuntos de datos de alta dimensionalidad.
- Robusto ante la existencia de ruido y anomalías en el conjunto de datos.
- El algoritmo tiene la capacidad de separar conjuntos de datos linealmente no separables mediante el uso de *kernels* complejos.

Contras:

- Excesiva dependencia a la parametrización (elección de núcleo, valores de C ...) para conseguir resultados óptimos.
- Requiere gran cantidad de memoria para su entrenamiento lo que resulta un impedimento a la hora de trabajar con grandes volúmenes de datos.

3.3 Netflow e IPfix

Netflow [21] es un protocolo de red desarrollado por Cisco Systems en 1996 como alternativa a SNMP a la hora de recopilar información sobre los flujos TCP/IP de una red. Capacita a los dispositivos para recolectar la información referente al tráfico que los atraviesa y define como dicha información es exportada para su posterior tratamiento.

Netflow es un protocolo propietario que se encuentra disponible en *routers* y *switches* de Cisco, Juniper, Enterasys y otros fabricantes. Existen variaciones en sus funcionalidades en función de la versión, siendo la v5 la v9 [22] las más estandarizadas.

La información del tráfico capturada puede variar de una versión a otra, pero los campos más habituales que se almacenan de cada uno de los flujos son los siguientes: fecha y hora de comienzo y finalización del flujo, dirección IP de origen y destino, puerto origen y destino, tipo de protocolo, interfaz lógica de entrada y salida, tipo de servicio, información de enrutamiento, bytes transferidos...

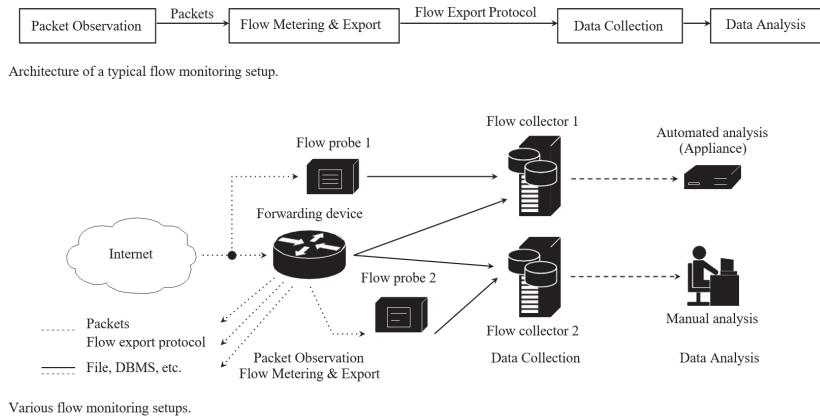


Figura 3.7: Arquitectura y configuración de un sistema de captura de flujos

Las etapas de una arquitectura de monitorización de flujo (ver figura 3.7) como puede ser la implementada a través de Netflow son las siguientes:

- *Packet Observation*: consiste en el proceso de capturar paquetes de la red y preprocesarlos. Esta fase es de una gran importancia ya que gran parte del rendimiento de la solución se basa en una correcta instalación y configuración de los dispositivos de captura. Durante esta fase se definen parámetros como la metodología de muestreo.
- *Flow Metering Export*: durante esta etapa los paquetes capturados se agregan a partir de unas propiedades comunes en un flujo determinado. La agregación se realiza basándose en determinados elementos de información que definen el diseño del flujo. Cada nueva entrada de flujo se almacena en una caché. El flujo se dará por completado según diferentes criterios que van desde una perspectiva temporal o de capacidad de la caché que lo almacena. Una vez terminada la agrupación los flujos se exportan a través de un datagrama del protocolo de exportación de flujo empleado.

Las dos primeras etapas pueden combinarse en un único dispositivo llamado *flow probe*.

- *Data Collection*: en esta etapa se recepcionan, almacenan y procesan los flujos generados por la etapa anterior. Los *flow collector* son los encargados de estas funciones.

Son tareas comunes a los *flow collector* la comprensión de los datos, la agregación , la anonimización y el filtrado.

- *Data Analysis*: constituye la última etapa en la cual se realizan tareas de correlación y agregación. Es el momento de definir en qué áreas podemos aplicar el análisis de los datos obtenidos a través de las diferentes tareas que podemos implementar en esta etapa: creación de perfiles de tráfico, clasificación y caracterización, anomalía y detección de intrusos y búsqueda de datos para fines forenses u otros fines de investigación.

3.3.1 IPfix (*IP Flow Information Export*)

Paralelamente al desarrollo de NetFlow, la IETF decidió en 2004 estandarizar un protocolo de exportación de flujo, y nació de ese modo el protocolo IPFIX [23]. NetFlow v9 fue elegido como la base del nuevo protocolo sin embargo, IPFIX no es solo una versión no propietaria de NetFlow v9 ya que IPFIX contiene características nuevas muy interesantes. Al no ser propietario es mucho menos rígido lo que le permite por ejemplo introducir fácilmente datos no estándar en el cuerpo del mensaje proporcionando por ello la posibilidad de mostrar más información.

3.4 IDS y NAT

Los sistemas de detección de intrusiones (IDS) son un tipo de componente de seguridad concebido para proteger redes o sistemas informáticos; vigilar el tráfico examinando redes y puertos y analizando paquetes de datos. El éxito de estos se basa en la habilidad que tengan para detectar el tráfico malicioso mediante las firmas u otras metodologías. Estas detecciones o alarmas pueden ser atómicas (constituidas por un paquete individual determinado) o compuestas (paquetes múltiples). Los IDS son capaces de distinguir entre dos métodos diferentes de intrusiones en la red:

- Basados en *host*, los cuales protegen estructuras informáticas centralizadas instalando agentes de supervisión que filtran el tráfico.
- Basados en la red, los cuales no comprueban datos sino paquetes IP por lo cual se vinculan normalmente al cortafuegos utilizado.

Los IDS actuales son sistemas híbridos que combinan ambos enfoques monitorizando datos, haciendo un análisis de los mismos en tiempo real y elaborando un informe de resultados según el riesgo potencial. Es fundamental que el análisis garantice la funcionalidad del sistema diferenciando de manera muy precisa entre los usos indebidos del mismo y la detección

de anomalías. Los IDS más conocidos y utilizados hoy en día son Suricata [24] y Snort [25], ambos gratuitos y *open source*.

Para comprender la diferencia entre IDS y las soluciones actuales de análisis de tráfico de red (NTA) [26], es importante comprender cómo ha cambiado el perímetro de la empresa, el tráfico de red y los atacantes. La respuesta que da NTA es fruto de aunar las diferentes técnicas (ML, IA, heurísticas, análisis de comportamiento...) y aplicarlas conjuntamente en tiempo real para así poder detectar las anomalías. Cada metodología implementa la seguridad y se complementan entre ellas logrando resultados más eficientes. NTA [27] fue creado y definido por primera vez por Gartner como una categoría emergente de soluciones de seguridad que utiliza las comunicaciones de red como la fuente de datos fundamental para detectar e investigar amenazas de seguridad y comportamientos anómalos o maliciosos dentro de esa red. NTA proporciona una visibilidad completa de toda la red para obtener información sobre las operaciones y el rendimiento de la misma.

Los ataques ciberneticos son cada vez más sofisticados por lo cual los IDS se han vuelto insuficientes porque tan solo se basan en la implementación de perímetros de red e identificación de las firmas. NTA supone un paso más en las técnicas de seguridad. Es resultado de los avances en la computación en la nube, la minería de datos y la IA y por ello puede prevenir y responder a ataques más elaborados.

3.4.1 La anatomía de un NTA

Una plataforma NTA generalmente consta de los siguientes elementos clave, algunos de ellos muy similares a los empleados en la captura de flujos:

- Colector de tráfico: recopila, analiza, normaliza y envía diferentes tipos de tráfico de red, como registros de tráfico y *pcaps*, al módulo de almacenamiento de datos. Los recolectores generalmente se implementan en modo de derivación para minimizar el impacto en las operaciones comerciales normales.
- Almacenamiento de datos de tráfico: los datos de tráfico se almacenan en una o múltiples bases de datos de manera centralizada o distribuida. Los registros de tráfico o metadatos se pueden consultar y buscar de manera flexible y eficiente. Por lo general, los datos se almacenan durante un período prolongado de tiempo. Para que un NTA sea efectivo, es crucial recopilar registros de tráfico para toda la red protegida durante períodos de tiempo suficientes.
- Motor de análisis de tráfico: existen muchas técnicas analíticas que pueden aplicarse, desde análisis estadísticos simples hasta algoritmos basados en aprendizaje automático mucho más complicados. El objetivo es identificar las aplicaciones y servicios cuyos patrones de tráfico exceden los umbrales de las líneas base establecidas.

- Módulo de salida: a medida que se generan los resultados del análisis de tráfico, los registros y las alertas, se presentan visualmente en la interfaz de usuario para que se puedan impulsar políticas de *firewall*, bloquear *hosts* sospechosos o realizar control de tráfico asociado con *hosts* comprometidos.

La tecnología NTA proporciona una herramienta eficaz y poderosa para obtener información sobre el tráfico de red y aplicaciones en tiempo real. Esto es fundamental para detectar amenazas posteriores a la violación, así como también aquellas actividades no autorizadas desde dentro de la red corporativa, ya sea intencionalmente o no.

3.5 ¿Por qué DataBricks?

Paso previo a explicar la elección de Databricks como espacio para desplegar nuestra implementación en un entorno distribuido vamos a exponer brevemente los conceptos principales de la plataforma Apache Spark [28], una de las múltiples soluciones del mercado que nos permite orquestar una infraestructura distribuida.

Apache Spark consiste en un sistema de computación distribuida de software libre que permite el uso simultáneo de un conjunto de computadores para procesar grandes conjuntos de datos. Spark está compuesto por los siguientes componentes:



Figura 3.8: Componentes de Apache Spark.

- Spark SQL: introduce un concepto de abstracción de datos llamado SchemaRD, que proporciona soporte para datos estructurados y semi-estructurados.
- Spark Streaming: es la capa encargada del análisis de datos en tiempo real.
- MLlib: es la capa de aprendizaje automático que proporciona el framework de aprendizaje automático donde se pueden encontrar multitud de algoritmos de clasificación, regresión, análisis *cluster*...

- GraphX: es la capa de procesamiento gráfico. Al basarse en RDDs inmutables, los gráficos no permiten actualizarse.
- Core: es el núcleo donde se apoya toda la arquitectura y una de sus principales características es el uso de *Resilient data distributed* (RDD).

RDD es la unidad de datos fundamental en Apache Spark. Un RDD es una colección distribuida e inmutable de elementos de datos. Cada conjunto de datos en RDD se divide en particiones lógicas, que se pueden calcular de forma paralela en diferentes nodos del *cluster*. Los RDD también son tolerantes a fallos. Todo esto permite que cada máquina del *cluster* pueda realizar una parte de la tarea, pero entre todas las realizan de una manera global proporcionando escalabilidad horizontal y tolerancia a los fallos.

Las aplicaciones para Spark se ejecutan como un grupo independiente de procesos en *clusteres*, coordinados por el objeto *SparkContext*, pudiendo conectarse a gestores de *cluster* que son los encargados de asignar recursos en el sistema. Una vez conectados, Spark puede encargar que se creen ejecutores encargados de la computación en los nodos del *cluster*.

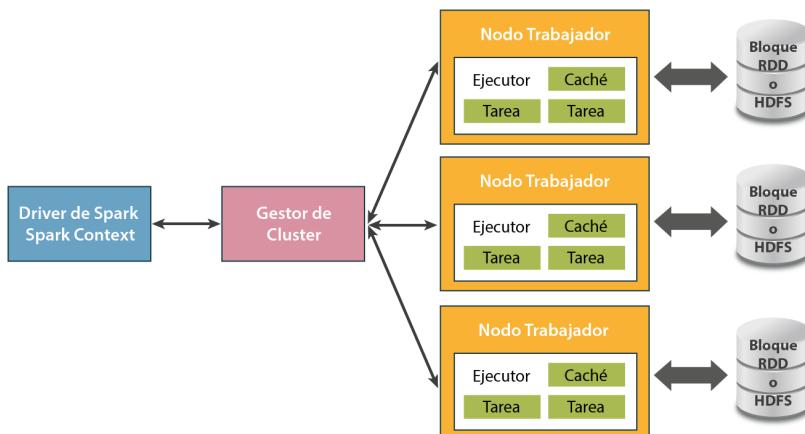


Figura 3.9: Funcionamiento de Apache Spark.

Databricks es un *workspace* basado en la plataforma distribuida Apache Spark. Su gran aceptación en la comunidad de minería de datos se debe a que, a través de sus *notebooks*, herramientas de desarrollo donde implementar las funcionalidades, y corriendo Apache Spark es posible realizar todo el ciclo de desarrollo de un proyecto de ML de una manera fácil e intuitiva. Esto permite, al igual que en Weka, que la curva de aprendizaje inicial sea muy suave. Databricks permite desplegarse tanto en Azure como en AWS y dispone de una herramienta de monitorización de la infraestructura integrada como es Ganglia. También proporciona de forma nativa herramientas de visualización que permiten mostrar gráficos de manera rápida y sencilla o crear *dashboard* más complejos según las necesidades. Se podría haber optado por

usar la infraestructura de AWS y crear desde 0 la arquitectura de *clusters* de Apache Spark, pero debido al alcance del proyecto se optó por simplificar este apartado haciendo uso de todo el potencial que ofrece Databricks.

3.6 ¿Por qué Python/PySpark?

El uso de Python como lenguaje de programación utilizado para las tareas relacionadas con la ingeniería de características está motivado debido a su gran implantación en el ámbito del ML [29]. Esto se debe a ser un lenguaje muy orientado y amigable en el tratamiento de los datos. La existencia de paquetes como Pandas y librerías como Scikit-learn apoyan estas consideraciones. También se ha tenido en cuenta la similitud entre Python y Pyspark para la elección final. Pyspark es una de las alternativas disponibles en el entorno de Apache Spark para la implementación de funcionalidades en el sistema distribuido al que da cobijo. Entre las características que podemos destacar de Pyspark están el soporte a sintaxis SQL y la posibilidad del uso distribuido de las librerías de Pandas y Scikit-learn del propio Python.

3.7 ¿Por qué Weka?

Weka es un *framework* para ML y *deeplearning* con un gran recorrido y un gran uso por parte de la comunidad de ML. Entre sus virtudes nos encontramos con que posee una gran documentación y una amplia bibliografía disponible. Su carácter *open source* y la gran comunidad que posee detrás le permiten seguir estando actualizado y que sus usuarios puedan mejorar o implementar nuevas funcionalidades a través del lenguaje Java que lo soporta. El motivo mayor para su elección ha sido que es un producto *all in one*. Cubre todos los aspectos necesarios en un proyecto de ML y da la posibilidad de su uso a través de GUI o CLI. La disponibilidad de una interfaz gráfica contribuye a tener una curva de aprendizaje sencilla.

3.8 Planificación

En el proyecto hemos escogido diferenciar entre la metodología ligada al proyecto, su planificación, y la metodología ligada a las tareas que lo componen. Para el primer punto hemos escogido *Scrum* como metodología de planificación.

3.8.1 *Scrum*

Scrum [30] es un modelo de desarrollo ágil que se caracteriza por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.

- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos autoorganizados, que en la calidad de los procesos empleados.
- Solapar las diferentes fases del desarrollo, en lugar de realizarlas una tras otra en un ciclo secuencial o de cascada.

Con el fin de implementar sus características, la metodología *Scrum* hace uso de los siguientes elementos (figura 3.10):

- Roles:
 - Dueño del producto: es la persona responsable de lograr el mayor valor de producto para los clientes, usuarios y resto de implicados.
 - Equipo de desarrollo: son el grupo o grupos de trabajo que desarrollan el producto.
 - *Scrum Master*: es el responsable del cumplimiento de las reglas de un marco de *Scrum* técnico, asegurando que se entienden en la organización, y se trabaja conforme a ellas. Proporciona la asesoría y formación necesaria al propietario del producto.
- Eventos:
 - *Sprint*
 - Reunión de planificación
 - *Scrum* diario
 - Revisión de *sprint*
 - Retrospectiva de *sprint*
- Artefactos:
 - Pila de producto
 - Pila de *sprint*
 - Incremento

La pieza clave del proceso iterativo en el que se basa *Scrum* es el *sprint*. Se denomina *sprint* a cada ciclo o iteración de trabajo que produce una parte del producto terminada y funcionalmente operativa (incremento). Se recomienda que la duración de cada *sprint* no exceda de un mes.

Se produce un incremento en cada iteración, aunque se puede incurrir en una excepción en el primer *sprint*. Cuando esto ocurre al *sprint* se le suele denominar *sprint 0*. Un ejemplo

CAPÍTULO 3. ENTENDER EL NEGOCIO

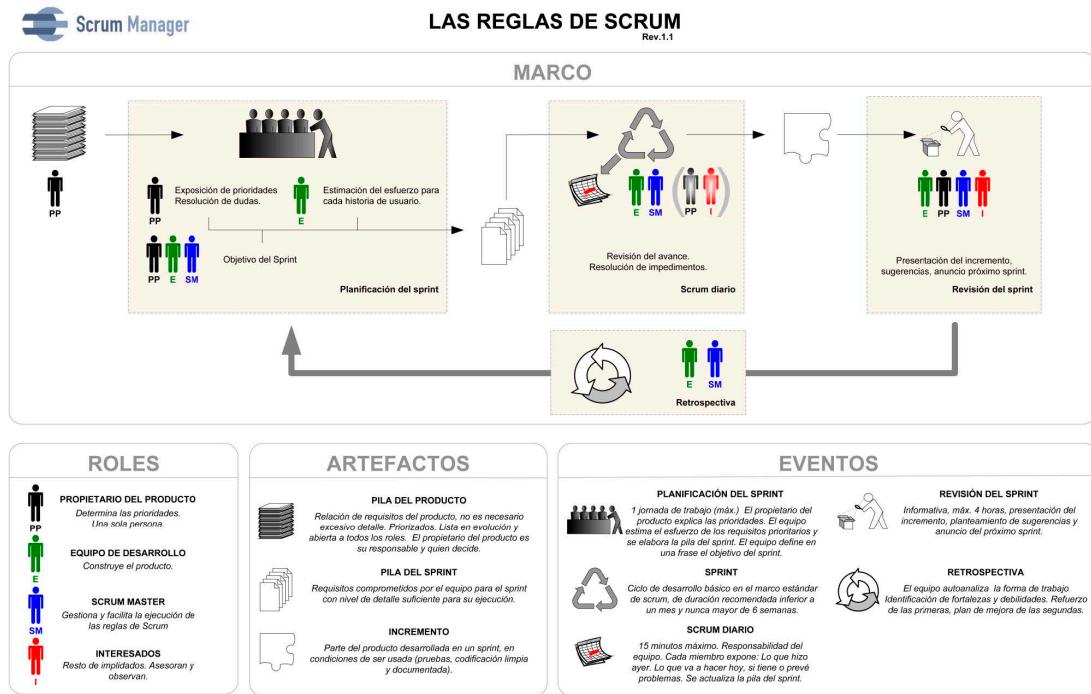


Figura 3.10: Marco Scrum técnico

son aquellos proyectos en los que son necesarios trabajos previos para conocer la tecnología o herramientas necesarias para la consecución del resto del proyecto.

Para construir cada incremento se han de realizar las historias de usuario previamente seleccionadas de la pila de producto para el *sprint*. La pila del producto registra los requisitos vistos desde el punto de vista del cliente, está formada por la lista de funcionalidades o historias de usuario que desea obtener el dueño del producto ordenadas por la prioridad que el mismo da a cada una. Las historias de usuario son utilizadas en los métodos ágiles para la especificación de requisitos, son una descripción breve de una funcionalidad software tal y como la percibe el usuario.

Estas reflejan las funcionalidades, pero no las tareas que el equipo de desarrollo necesita para realizarlas por lo cual cada *sprint* posee su propia pila llamada pila del *sprint*. Dicha pila está formada por la lista de tareas resultantes de la descomposición por parte del equipo de las historias de usuario en unidades de trabajo adecuadas para gestionar y seguir el avance de su ejecución.

Por encima de tareas e historia de usuarios nos encontramos las épicas, que se caracterizan por tener un tamaño demasiado grande para ser completadas de una sola vez o en un solo *sprint*.

Respecto a la metodología de las tareas, haremos uso de una de las más extendida en los procesos de minería de datos: CRISP-DM (cap. 2.1). Fruto de esta elección, las historias

de usuario de las que se compone nuestra pila del producto, y algunas de las tareas que las desglosan, son las diferentes tareas recogidas en el flujo de trabajo de CRISP-DM (figura 2.2). En consecuencia, la pila de producto resultante que dará lugar a los diferentes *sprints* será:

- Entender el negocio.
- Modelo de regresión logística.
- Modelo de *Random Forest*.
- Modelo de SVM.
- Despliegue.
- Redacción de la memoria.

Tras definir las historias de usuarios y realizar la estimación del esfuerzo necesario para llevarlas a cabo decidimos que la duración de los *sprints* del proyecto será de 3 semanas, con semanas de cinco días de trabajo. Hemos englobado todas las historias de usuario dentro de una épica denominada TFG. La duración estimada de la consecución de la épica es de 90 puntos de historia, entendiéndose como un punto de historia el trabajo realizado durante una jornada de 5 horas. El desglose de los diferentes *sprints*, uno por cada una de las historias de usuario de la pila del producto, y sus pilas asociadas están representados en las tablas incluidas en los anexos A.1.

Capítulo 4

Entender los datos

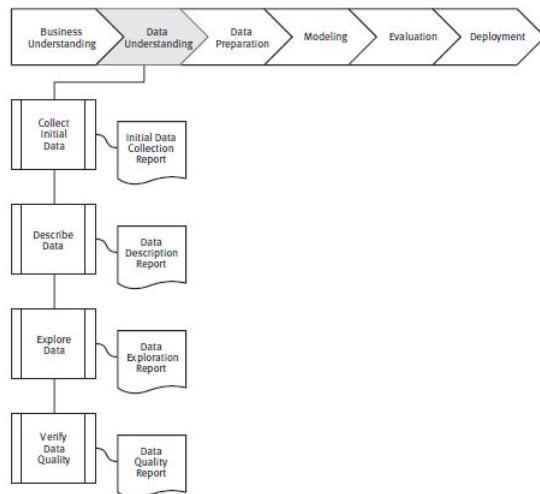


Figura 4.1: Fase CRISP-DM: Entender los datos.

4.1 Descripción

El *dataset* seleccionado fue creado expresamente para realizar trabajos referentes a la detección de anomalías, en concreto para la evaluación de IDSs. Este *dataset* es el UNB ISCX IDS 2012 [31]. El *dataset* se encuentra etiquetado en su totalidad e integra los flujos generados a partir de los archivos *.pcap* que forman la captura del tráfico. Este último punto ha sido fundamental ya que nos ha permitido prescindir de la captura de flujo y su posterior etiquetado, tarea no trivial en su ejecución. La creación de los flujos no es determinista e incluso con los *.pcap* y la configuración del exportador/colector adecuada, no podemos garantizar que los flujos creados fuesen idénticos a los incluidos en el *dataset*, lo que provocaría la necesidad de generar las etiquetas de forma manual.

Las características principales del *dataset* [32] son las siguientes:

- Tráfico sintético que incluye varios casos de ataque en un escenario realista (figura 4.2)
 - Intento de intrusión desde dentro de la red
 - HTTP DoS
 - IRC Botnet DoS
 - Ataque de fuerza bruta sobre SSH
- Etiquetado y captura completa
- Flujos bidireccionales incluidos
- Gran tamaño

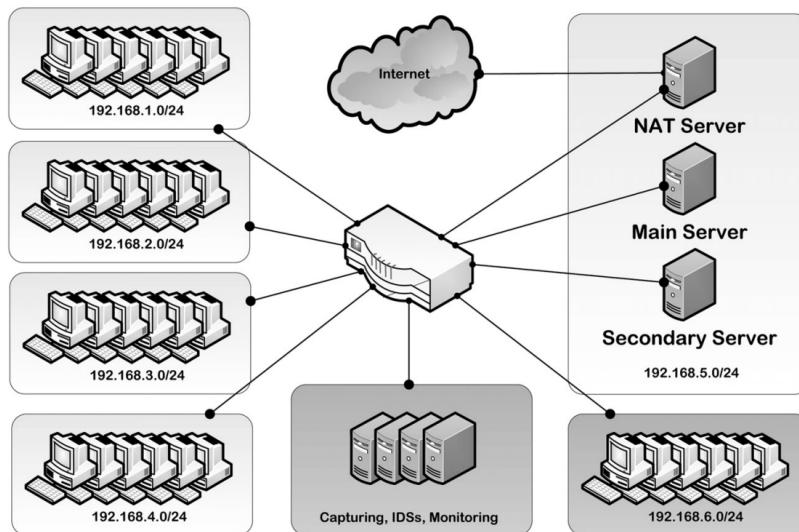


Figura 4.2: Escenario del *dataset* UNB ISCX IDS 2012

Sin embargo, presenta algunos defectos [33] debidos a la antigüedad del *dataset* y a la metodología de generación del tráfico sintético. En primer lugar, no representa el estado actual del tráfico, a pesar de que el tráfico HTTPS supone el 70% del tráfico en la actualidad, este no aparece reflejado en el *dataset*. En segundo lugar, el tráfico maligno generado no sigue una distribución realista.

4.2 Recolección

La confección de un *dataset* completo y de calidad a través de la captura de flujos en un entorno real o virtualizado se valoró como excesivamente compleja para el alcance final del proyecto. En su lugar se realizó una prueba de concepto consistente en la instalación y prueba

[34] de los elementos necesarios para realizar la captura de flujos y se seleccionó como *dataset* de trabajo el *dataset* descrito en la sección anterior.

La fase de recolección se limitó a la importación de los datos desde los ficheros *.XML* incluidos en el *dataset*, sólo en aquellos días en los que existía algún tipo de ataque. Durante la importación se aprovechó para incluir un nuevo atributo denominado *typeAttack* (tipo de ataque) con la intención de ser usado posteriormente durante los procesos de validación cruzada 6.2.2.

4.3 Exploración

El *dataset* consta inicialmente de 22 atributos. En el anexo A.2 se detalla cada uno de ellos.

RangeIndex:	1416201 entries, 0 to 1416200
Data columns (total 22 columns):	
Tag	1416201 non-null object
appName	1416201 non-null object
date	1416201 non-null object
destination	1416201 non-null object
destinationPayloadAsBase64	620061 non-null object
destinationPayloadAsUTF	620008 non-null object
destinationPort	1416201 non-null object
destinationTCPFlagsDescription	1366947 non-null object
direction	1416201 non-null object
protocolName	1416201 non-null object
source	1416201 non-null object
sourcePayloadAsBase64	698046 non-null object
sourcePayloadAsUTF	784626 non-null object
sourcePort	1416201 non-null object
sourceTCPFlagsDescription	1413843 non-null object
startDateTime	1416201 non-null datetime64[ns]
stopDateTime	1416201 non-null datetime64[ns]
totalDestinationBytes	1416201 non-null int64
totalDestinationPackets	1416201 non-null int64
totalSourceBytes	1416201 non-null int64
totalSourcePackets	1416201 non-null int64
typeAttack	1416201 non-null object

Figura 4.3: Información general del *dataset*

En la fase de importación, durante la cual se asignó el tipo de cada atributo, se tomó la decisión de solo considerar numéricos aquellos atributos en los que su valor tuviese un significado coherente dentro de los modelos a construir. Es por ello que los atributos que hacen referencia a los puertos no se hayan tomado como tal. De estos atributos podremos

extraer conocimiento con la creación de unos nuevos, fruto de la agregación por rango de sus valores.

La primero que observamos en la fig. 4.3 es que nos encontramos con un *dataset* con cerca de un millón y medio de muestras. A simple vista podríamos pensar que nuestro principal problema podría ser tener que trabajar con una gran número de muestras durante todo el proceso, pero veremos que debido al desbalanceo existente entre las dos clases etiquetadas (95.28% vs 4.72%), fig. 4.4, el número final que usaremos, véase sección 5.3, se reducirá al 10%.

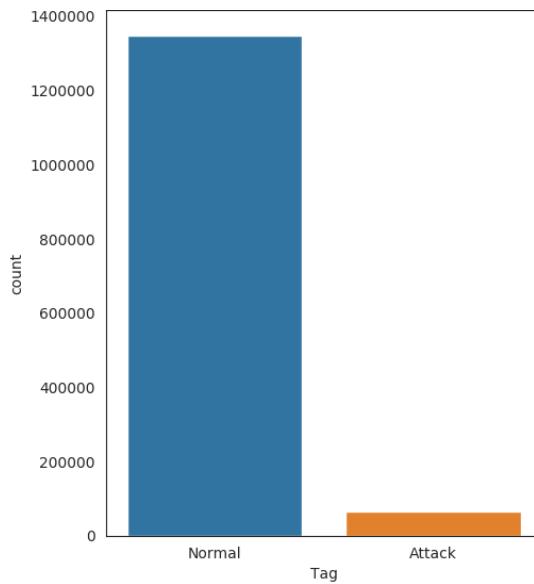


Figura 4.4: Distribución de la clase objetivo

Una de las primeras tareas en la exploración de nuestros datos debería ser comprobar la completitud de nuestro *dataset*. En la figura 4.3 apreciamos que los atributos listados a continuación no tienen el número de valores esperado (**1416201**)

- *destinationPayloadAsBase64* - **620061**
- *destinationPayloadAsUTF* - **620008**
- *destinationTCPFlagsDescription* - **1366947**
- *sourcePayloadAsBase64* - **698046**
- *sourcePayloadAsUTF* - **784626**
- *sourceTCPFlagsDescription* - **1413843**

Para abordar este análisis primero nos centraremos en los atributos *TCPFlag*, usaremos *destinationTCPFlagsDescription* como ejemplo ya que el comportamiento de *sourceTCPFlags-Description* es análogo.

F, S, P, A	756548
N/A	301936
F, S, R, P, A	85214
S, P, A	64051
NaN	49254
F, A	45283
R, A	30113
P, A	19564
R	14574
F, S, A	10684
F, P, A	10616
S, A	8460
S, R, P, A	4749
S, R, A	3697
F, R, A	3017
F, R, P, A	2979
F, S, R, A	2593
A	2548
S	140
R, P, A	140
F, S, P, A, Illegal18	16
R, A, Illegal17, Illegal18	9
F, S, R, P, A, Illegal18	6
R, A, Illegal17	3
S, P, A, Illegal18	2
S, R, A, Illegal18	2
F, P, A, Illegal18	1
R, A, Illegal18	1
R, Illegal18	1

Figura 4.5: Valores presentes en el atributo *destinationTCPFlagsDescription*

El número de valores faltantes se debe a los nulos, nombrados aquí como *NaN*. Realizamos ahora el estudio que nos permitirá tomar las medidas correctoras adecuadas para solucionar el problema: eliminar las entradas o completar los valores.

Comparamos los diferentes valores con los establecidos en la especificación [35] y concluimos que se pueden referir a aquellas muestras de tráfico TCP que no tengan ningún *flag* presente.

Comprobamos que esto es correcto y que además los casos en los que no debe existir *flags*, como en el tráfico UDP, ICMP..., se han etiquetado como N/A. Durante la fase de limpieza reasignaremos el valor *NaN* al valor *Blank*

tcp_ip	49254
--------	-------

Figura 4.6: Valores con *flag NaN* en tráfico TCP en el atributo *destinationTCPFlagsDescription*

udp_ip	294210
icmp_ip	7617
igmp	58
ip	51

Figura 4.7: Valores con *flag N/A* en tráfico TCP en el atributo *destinationTCPFlagsDescription*

Tras comprobar que el valor *Illegal* 4.5 no es un valor válido en la especificación y debido a que el número de flujos afectados, alrededor de 50, frente al total, cerca de un millón y medio es despreciable, optamos por eliminar los flujos durante la fase de limpieza.

Estudiaremos ahora el segundo grupo de atributos con valores faltantes:

- *sourcePayloadAsBase64*
- *sourcePayloadAsUTF*
- *destinationPayloadAsBase64*
- *destinationPayloadAsUTF*

En esta ocasión los valores *NaN* hacen referencia a aquellos flujos que no tienen carga útil (*payload*), lo cual es un escenario esperado. Tras realizar este análisis hemos decidido prescindir de la información del contenido del *payload*, pero sí que nos parece importante conservar la información de su presencia o ausencia. Por ello crearemos durante la fase de transformación los atributos *sourcePayload* y *destinationPayload* que contendrán dicha información y eliminaremos los atributos originales como parte de la fase de limpieza.

Por último, vamos a comprobar si los valores referentes al atributo *appName* hacen referencia a servicios comúnmente atacados. Este análisis nos permitiría en la fase de selección reducir el número de características del *dataset*.

Para la comparativa utilizaremos la herramienta *nmap* que nos muestra la *n*-clasificación de los puertos más escaneados [36]. Somos conscientes de que es una aproximación simple asumir que la probabilidad de ataque sobre un servicio solo esté determinado por el puerto escaneado, pero de todos modos nos parece un acercamiento lo suficientemente bueno para realizar una posterior reducción de la dimensionalidad del *dataset*.

```
sudo nmap -sTU --top-ports 20 localhost -v -oG -
# Nmap 7.60 scan initiated Sun Aug 25 03:50:39 2019 as: nmap -sTU
--top-ports 20 -v -oG - localhost
# Ports scanned: TCP(20;21-23,25,53,80,110-111,135,139,143,443,445
,993,995,1723,3306,3389,5900,8080)
UDP(20;53,67-69,123,135,137-139,161-162,445,500,514,520,631,1434
,1900,4500,49152) SCTP(0;) PROTOCOLS(0;)
```

Figura 4.8: Comando *nmap* para mostrar el top 20 de puertos escaneados de los protocolos TCP y UDP

```
[ 'FTP', 'SSH', 'Telnet', 'SMTP', 'DNS', 'DNS-Port', 'PeerEnabler',
'MiscApplication', 'TFTP', 'HTTPWeb', 'HTTPImageTransfer',
'WebMediaDocuments', 'WebMediaAudio', 'WebFileTransfer',
'BitTorrent', 'WebMediaVideo', 'POP', 'SunRPC', 'NTP', 'RPC',
'NetBIOS-IP', 'WindowsFileSharing', 'IMAP', 'SNMP-Ports',
'SecureWeb', 'IPSec', 'SSL-Shell', 'Unknown_TCP', 'MS-SQL',
'PPTP', 'Unknown_UDP', 'SSDP', 'MiscApp', 'MSN-Zone',
'MSTerminalServices', 'VNC' ]
```

Figura 4.9: Servicios de nuestro *dataset* coincidentes con el top *nmap*

De los 107 valores de los que dispone el *dataset*, solo 36 corresponden a servicios ligados al top 20 de puertos escaneados. Con esta información, durante la etapa de reducción de características, eliminaremos aquellos que no estén en la lista.

Con toda la información recopilada durante esta fase sobre el dataset de estudio podemos empezar a acometer las tareas que nos permitan preparar los datos como paso previo al modelado de los algoritmos de ML seleccionados.

Capítulo 5

Preparar los datos

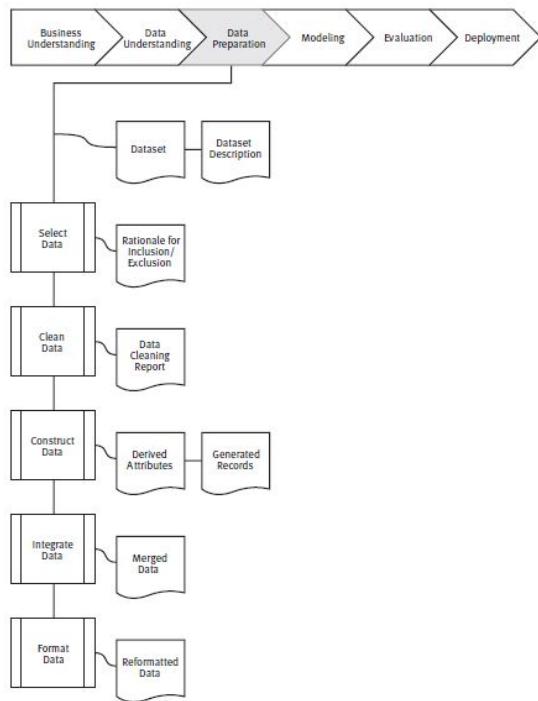


Figura 5.1: Fase CRISP-DM: Preparar los datos.

5.1 Limpieza

La existencia de datos anómalos, también llamados atípicos, en el *dataset* puede provocar que los modelos creados no se adecúen a la realidad del problema que queremos resolver. Esta eliminación no es siempre necesaria ya que no todos los algoritmos son sensibles a su presencia. Como ya hemos visto en el capítulo 3, de los algoritmos de nuestro estudio solo regresión logística 3.4 presenta este problema.

Ahora bien, la eliminación supone la ventaja de disminuir la carga computacional y el volumen de datos durante las fases de entrenamiento y evaluación del modelado. Esta es la razón de realizar también el cambio en los *dataset* que usaremos para los algoritmos de *Random Forest* y regresión logística

Un reto al abordar esta tarea es no olvidar que justamente nuestro proyecto se trata de identificar datos anómalos ligados a ataques, por lo que no tendría sentido eliminarlos de nuestro dataset antes del modelado. En este caso entendemos como datos atípicos aquellos que lo son, pero dentro de la etiqueta Normal de nuestro atributo objetivo. La idea es eliminar valores anormales dentro del tráfico normal que puedan llegar a desvirtuar el entrenamiento del algoritmo.

En lugar de realizar una búsqueda y limpieza usando todo el *dataset*, hemos optado por realizar una agrupación alrededor del atributo *appName*. Esto es debido a que consideramos que la caracterización según el servicio es representativa de la tipología del tráfico que engloba.

Existen diferentes maneras de abordar la limpieza [37] de los valores atípicos pero para nuestro estudio hemos escogido solo dos de ellas.

5.1.1 Diagrama de caja

Nos encontramos en un método basado en el concepto estadístico de rango intercuartílico (IQR) y en su representación visual de la mano de los diagramas de cajas 5.2. Su elección se debe a que es un método sencillo y ampliamente conocido.

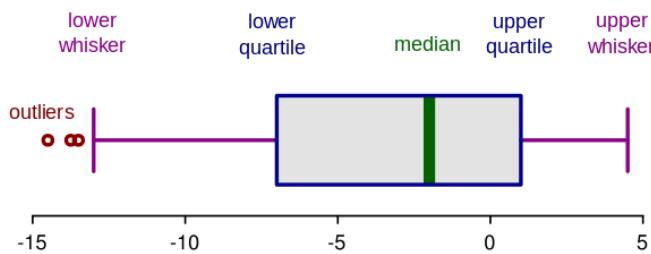


Figura 5.2: Diagrama de cajas y datos anómalos(*outliers*)

IQR hace referencia a la distancia entre el 1º y 3º cuartil y permite definir la posición ($\pm 1.5 * IQR$) de los bigotes de la caja. Aquellos datos por debajo del bigote menor y los que se encuentren por encima del bigote mayor son considerados atípicos

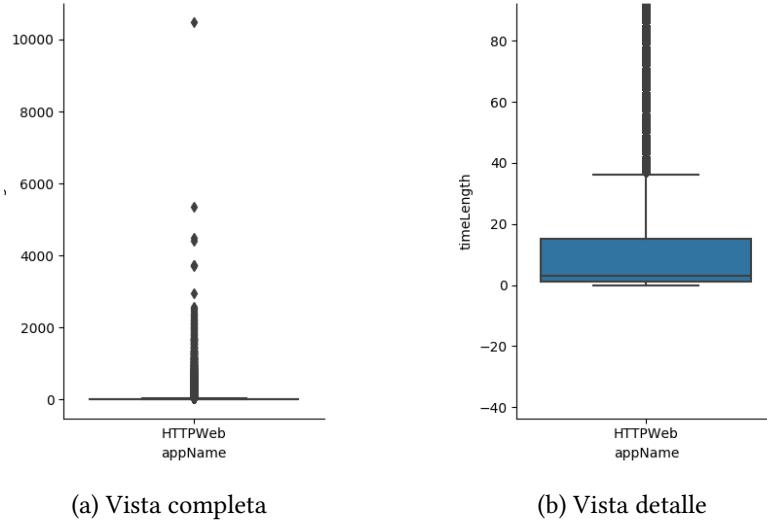


Figura 5.3: Diagrama de caja del atributo *timelength* agrupado por *HTTPWeb*

Las pruebas realizadas, documentadas dentro del código del repositorio Git A.3, demuestran un comportamiento muy agresivo eliminando alrededor del 26% de las muestras. Una cifra excesivamente elevada.

5.1.2 *Isolation forest*

Isolation forest [38] es un método de aprendizaje sin supervisión de la familia de los árboles de decisión. La característica principal que lo distingue de otros métodos de detección de valores atípicos es que en esta ocasión el esfuerzo del algoritmo es detectar los datos anómalos, no caracterizar los datos normales. La idea se puede ver en la figura 5.4 y en cómo son necesarias menos particiones para alcanzar el resultado final. Esta característica lo convierte en un excelente método en conjunto de una alta dimensionalidad.

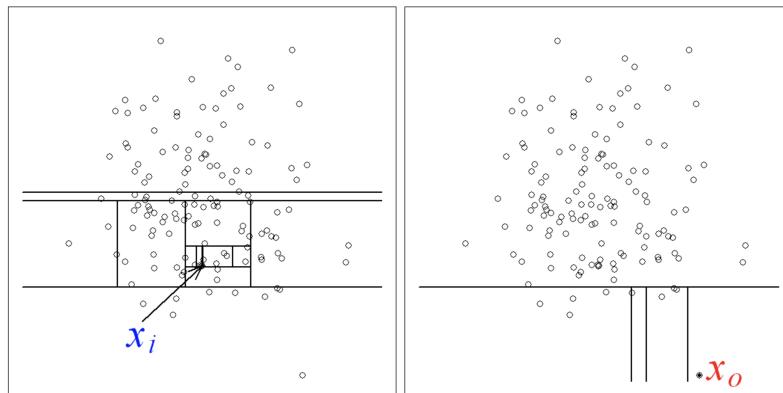


Figura 5.4: Diferencia entre identificar muestras anómalas o identificar muestras normales

En la implementación del algoritmo *Isolation Forest* se puede estimar previamente el porcentaje de datos atípicos que creemos están presentes en las muestras. Tras analizar la distribución de los datos de nuestros *dataset* y con una aproximación no agresiva, hemos optado por fijar este número en un 5%.

	count	mean	std	min	50%	75%	80%	85%	90%	95%	96%	97%	98%	99%	max
HTTPWeb	478625.00000	28.54041	431.22879	0.00000	6.00000	17.00000	21.00000	29.00000	45.00000	75.00000	89.00000	106.00000	148.00000	303.00000	138707.00000
HTTPImageTransfer	453346.00000	15.88694	58.41711	0.00000	6.00000	11.00000	13.00000	18.00000	28.00000	50.00000	60.00000	74.00000	103.00000	179.00000	1515.00000
DNS	200766.00000	1.89058	3.60965	0.00000	1.00000	1.00000	2.00000	3.00000	6.00000	7.00000	9.00000	10.00000	13.00000	168.00000	
Unknown_UDP	62195.00000	12.02587	326.91937	0.00000	1.00000	4.00000	4.00000	4.00000	5.00000	8.00000	9.00000	9.00000	10.00000	12.00000	28971.00000
SecureWeb	44108.00000	17.43518	64.83506	0.00000	7.00000	11.00000	14.00000	17.00000	24.00000	50.00000	67.72000	93.00000	114.00000	213.00000	3698.00000
NetBIOS-IP	20697.00000	1.33585	1.30301	0.00000	1.00000	1.00000	2.00000	3.00000	3.00000	3.00000	3.00000	3.00000	3.00000	4.00000	60.00000
Unknown_TCP	14062.00000	5.05774	129.73430	0.00000	3.00000	3.00000	3.00000	3.00000	3.00000	3.00000	6.00000	8.00000	11.00000	8345.00000	
WindowsFileSharing	12722.00000	7.09770	6.51964	1.00000	11.00000	11.00000	12.00000	17.00000	18.00000	19.00000	19.00000	21.00000	24.00000	26.00000	
POP	12292.00000	6.40408	18.27176	1.00000	8.00000	8.00000	8.00000	8.00000	8.00000	9.00000	9.00000	9.00000	9.00000	9.00000	1987.00000
BitTorrent	10136.00000	29.59905	416.48092	1.00000	5.00000	6.00000	6.00000	7.00000	9.00000	10.00000	12.00000	13.00000	36.65000	17066.00000	
FTP	9399.00000	135.96808	7513.89583	1.00000	4.00000	5.00000	17.00000	52.00000	53.00000	55.00000	60.00000	63.00000	115.04000	514794.00000	
ICMP	7373.00000	2.39251	3.30125	0.00000	1.00000	3.00000	4.00000	4.00000	4.00000	8.00000	8.00000	9.00000	11.00000	15.00000	122.00000
SMTP	7026.00000	9.52533	64.09860	1.00000	7.00000	7.00000	7.00000	8.00000	8.00000	9.00000	15.00000	20.00000	3997.00000		
IMAP	6858.00000	4.19612	15.92800	1.00000	1.00000	1.00000	1.00000	15.00000	17.00000	18.00000	19.00000	20.00000	21.00000	30.00000	1190.00000
WebMediaDocuments	3438.00000	80.78941	849.35308	0.00000	10.00000	30.00000	45.60000	81.00000	131.00000	277.15000	335.04000	426.78000	577.26000	973.42000	46990.00000
Flowgen	2069.00000	11.85839	212.83597	1.00000	1.00000	4.00000	4.00000	4.00000	5.00000	8.00000	8.00000	9.00000	10.00000	14.32000	6306.00000
SSH	1714.00000	25.79347	106.89029	1.00000	15.00000	21.00000	22.00000	24.00000	28.00000	36.35000	206.00000	212.00000	217.00000	330.00000	3369.00000
WebFileTransfer	526.00000	48.28707	90.13245	2.00000	11.00000	39.75000	50.00000	152.00000	253.75000	267.00000	287.25000	350.00000	425.25000	643.00000	
MiscApplication	365.00000	1.99726	14.46111	1.00000	1.00000	1.00000	1.00000	2.00000	2.80000	4.00000	4.00000	4.72000	5.36000	277.00000	
WebMediaVideo	245.00000	1083.82041	814.29512	6.00000	764.00000	1760.00000	1891.60000	2041.40000	2229.00000	2370.60000	2393.32000	2422.28000	2449.12000	2494.32000	6147.00000

Figura 5.5: Distribución de los valores *timelength* en el tráfico normal

Elegimos *Isolation Forest* para realizar la transformación total del *dataset* dando como resultado la eliminación del 4.7% del total de datos.

	totalDestinationBytes	totalDestinationPackets	totalSourceBytes	totalSourcePackets	timeLength
count	478625.00000	478625.00000	478625.00000	478625.00000	478625.00000
mean	58744.38696	45.55344	2086.79466	28.54041	21.87410
std	1188688.52514	820.57000	28097.48553	431.22879	59.24132
min	0.00000	0.00000	0.00000	0.00000	0.00000
50%	1337.00000	5.00000	479.00000	6.00000	3.00000
75%	28905.00000	24.00000	1228.00000	17.00000	15.00000
80%	35437.00000	29.00000	1504.00000	21.00000	24.00000
85%	56627.20000	44.00000	2127.00000	29.00000	45.00000
90%	93294.80000	70.00000	3125.00000	45.00000	77.00000
95%	163972.00000	138.00000	5835.00000	75.00000	103.00000
96%	174067.24000	160.00000	7575.00000	89.00000	117.00000
97%	207402.00000	208.00000	9372.00000	106.00000	149.00000
98%	332960.68000	242.00000	10686.52000	148.00000	179.00000
99%	710517.68000	505.00000	21888.40000	303.00000	209.00000
max	395599648.00000	275310.00000	8918861.00000	138707.00000	10474.00000

(a) Inicial

	totalDestinationBytes	totalDestinationPackets	totalSourceBytes	totalSourcePackets	timeLength
count	454693.00000	454693.00000	454693.00000	454693.00000	454693.00000
mean	18339.33303	16.37906	915.48581	12.19987	18.03790
std	35621.49751	25.73671	1079.43565	14.44794	38.70755
min	0.00000	0.00000	0.00000	0.00000	0.00000
50%	1128.00000	5.00000	457.00000	6.00000	2.00000
75%	23361.00000	20.00000	1123.00000	15.00000	13.00000
80%	29938.00000	24.00000	1269.00000	17.00000	19.00000
85%	38218.20000	31.00000	1567.00000	22.00000	34.00000
90%	60470.00000	47.00000	2223.80000	31.00000	67.00000
95%	102566.00000	76.00000	3212.00000	46.00000	100.00000
96%	115094.00000	81.00000	3479.00000	49.00000	102.00000
97%	122249.40000	91.00000	3859.00000	54.00000	111.00000
98%	144652.80000	104.00000	4473.00000	60.00000	153.00000
99%	169793.00000	132.00000	5220.00000	71.00000	185.00000
max	233902.00000	180.00000	14618.00000	103.00000	1467.00000

(b) Sin valores atípicos

Figura 5.6: Resultado del uso de isolation Forest en el tráfico normal *HTTPWeb*

5.2 Transformación

En esta fase [39, cap. 4] agrupamos los procesos encargados tanto de adecuar los atributos existentes a las necesidades del problema, como la creación de nuevos atributos, habitualmente derivados de los anteriores.

Algunas de estas transformaciones vienen dadas por el análisis y las acciones tomadas en las anteriores etapas, como es el caso de la creación de nuevos atributos para guardar las informaciones del *payload* 4.3. Otras están ligadas a la tipología del algoritmo sin las cuales no se puede realizar de forma correcta el entrenamiento, como ocurre con el escalado.

5.2.1 Creación

Creamos un nuevo atributo *timeLength* por dos motivos, primero porque será un valor con una información temporal mucho más comprensible que los datos de las fechas que lo crearon, segundo porque las fechas como tal han de ser eliminadas ya que se comportan como claves candidatas dando lugar a asociaciones únicas con la clase objetivo que desvirtuaría el modelo. De querer usarlas, habría que descomponerlas en atributos más generales como día de la semana, franja horaria...

Creamos nuevas variables derivadas de las que consideramos más significativas a la hora de caracterizar un flujo. Es más común referirse al tráfico en una red por sus tasas más que por sus acumulados.

- sourceByteRate
- destinationByteRate
- sourcePacketRate
- destinationPacketRate
- avgSourcePacketSize
- avgDestinationPacketSize

La creación más ambiciosa es el cálculo de la diferencia de cada uno de los atributos numéricos que caracterizan el tráfico frente a la mediana respecto al *appName*. La intención es aflorar el comportamiento de los atributos de cada flujo agregando por su *appName*. De esta manera podemos ver la desviación que se produce respecto a la mediana de ese atributo en particular. Empíricamente consideramos que un tráfico normal no debería separarse de la mediana de sus datos, más si tenemos en cuenta la cantidad de datos de los que disponemos para hacer ese cálculo y que la mayoría de ellos son datos normales. Usamos mediana en lugar de media ya que esta es un estadístico mucho más robusto a las anomalías.

Para el cálculo de la mediana no tenemos en cuenta los datos identificados anteriormente por *Isolation Forest* como anómalos. Solo se calculará con tráfico normal no atípico.

Tras finalizar la creación de los nuevos atributos, para poder realizar una comparación no sesgada entre las diferencias con las medianas entre los diferentes tipos de *appName*, es necesario realizar un proceso de escalado que explicaremos en el siguiente punto.

Como último paso de creación convertiremos los atributos categóricos multievaluados a tantas variables binarias como clases contengan. La implementación de esta técnica se denomina *Hot-Encoded*. Es un paso imprescindible en los métodos SVM y LR ya que solo trabajan con valores numéricos.

5.2.2 Escalado

Existen varios motivos para realizar el escalado. Puede ser como ocurre en los modelos basados en el cálculo de la distancia euclídea, en donde se quiere evitar que los valores altos tengan un mayor peso en el cálculo final del modelo o en otras situaciones cuando se necesita comparar atributos de un mismo tipo, pero con un diferencia muy grande en sus valores.

El primer caso lo podemos encontrar en la necesidad de escalar los valores para poder construir el modelo del algoritmo SVM y el segundo caso en el escalado necesario para que el atributo **DiffMedia* 5.2.1 fuese comparable entre cada uno de los tipos de tráfico por los que fue agregado.

Los diferentes métodos de escalado [40] tienen características que no les permite ser usados en todos los escenarios. No todos devuelven los valores entre un rango predefinido (*RobustScaler*) y otros se comportan mal ante la presencia de datos atípicos (*MinMaxScaler*)

Debido a que hemos realizado una limpieza previa de datos atípicos, podemos usar como tipo de normalización *MaxAbsScaler* aún a sabiendas que *RobustScaler* es un normalizador más robusto a los datos atípicos. Primamos la necesidad de que los datos se mantengan en un rango determinado para que puedan ser comparables las diferentes agrupaciones sobre las que se han calculado la diferencia de la mediana.

CAPÍTULO 5. PREPARAR LOS DATOS

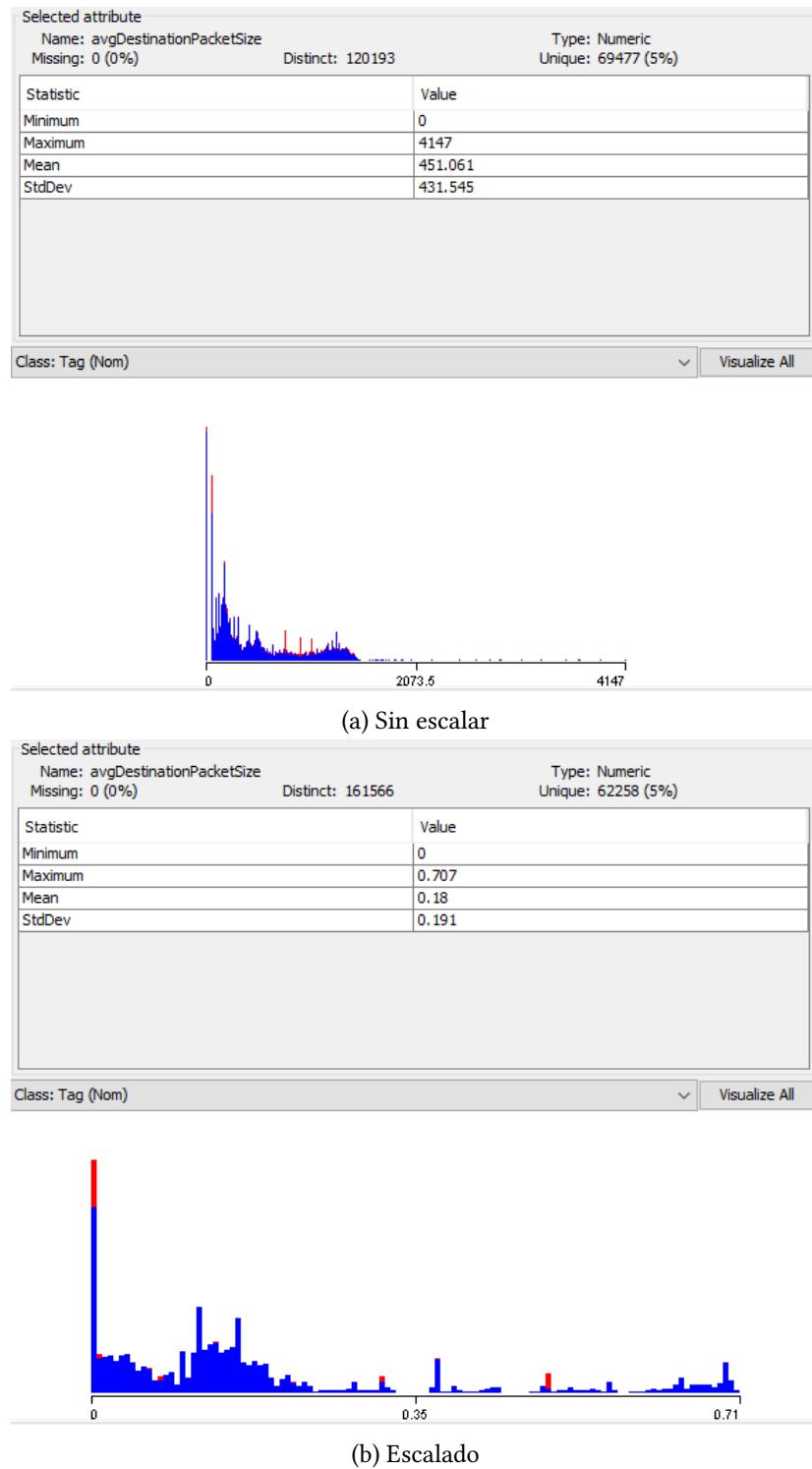


Figura 5.7: Cambios tras escalar en el atributo *avgDestinationPacketSize*

5.2.3 Discretización de valores

Consiste en la conversión de un valor numérico en un valor categórico. Entre sus múltiples usos están paliar ciertos problemas de interpretación de los valores, como en el caso de una relación no lineal, o reflejar la existencia de umbrales significativos en los valores.

Bajo este enfoque realizamos el desglose de los puertos en su agregación más habitual [41]:

- *well-known*: 0 – 1023
- *registered*: 1024 – 49151
- *ephemeral*: 49151 – 65535

En el caso de las direcciones IP nos encontramos con que sus valores numéricos no tienen una interpretación lineal, por lo que realizaremos una clasificación de destino y origen según el documento del *dataset* y reflejado en la captura del diseño del escenario 4.2:

- *invalid*
- *NATServer*
- *mainServer*
- *secondaryServer*
- *internal*
- *private*
- *external*

5.3 Selección

La selección de características es la denominación de la técnica que permite estudiar la relevancia de los atributos y seleccionar de forma autónoma el mejor subconjunto, entendiéndose el mejor generalmente como aquel que permite al modelo conseguir la mayor precisión. Es importante no confundirla con la reducción de la dimensionalidad, ya que en aquellas se pueden crear y transformar los atributos mientras que en esta solo es una simple selección.

En los proyectos de estas características, donde están involucrados *datasets* con muchos elementos y características, el consumo de recursos eficientes, memoria, cpu, red..., son factores muy importantes. La selección de características permite reducirlas optimizando el rendimiento en términos de precisión y carga computacional.

Antes de abordar la selección de características, nos queda solucionar el problema del desbalanceo que existe en la clase objetivo 4.3.

Cuando un conjunto de datos no está equilibrado, suele ocasionar modelos de ML que proporcionan un bajo rendimiento en términos de precisión, generalmente ocasionados por problemas de sobreestimación debido a la presencia de datos irrelevantes o redundantes. A pesar de que metodologías como *bootstrap* [39, cap. 17, p. 466] permiten lidiar con esta situación, el escenario óptimo es tener conjuntos de entrenamiento lo más balanceados posibles. El conocimiento está en los datos y es su calidad lo que define en gran medida el resultado final. En casos extremos se puede producir lo que se denomina paradoja de la precisión [42], en la que el modelo descarta directamente los casos de la categoría representada en menor medida, ya que la elección de la categoría mayoritaria siempre le va a reportar un valor de precisión óptimo. Existen diferentes aproximaciones para resolver esta problemática [43], pero en el escenario que tenemos, con un *dataset* de gran tamaño en el que la clase desbalanceada presenta alrededor de 70 mil muestras, la solución escogida ha sido realizar un inframuestreo [44] del conjunto de datos manteniendo una proporción 1:1.

La decisión tomada de realizar las acciones correctivas sobre el desbalanceo 4.3 al final del proceso, ha sido por considerar que eliminar el 90% de la información al inicio, perjudicaría al resto de etapas de ingeniería de características, sobre todo a aquellas ligadas a transformaciones basadas en cálculos estadísticos. Entendemos que los resultados obtenidos son más representativos con la totalidad de las muestras.

Corregido el desbalanceo entre las clases objetivo entramos de lleno en la fase de selección propiamente dicha. Existen principalmente dos metodologías para afrontarla, las basadas en filtros y las basadas en modelos.

- Los basados en filtro hacen uso principalmente de técnicas estadísticas aplicadas sobre los propios datos. Se caracterizan por tener un coste computacional reducido.
- Los basados en modelo hacen uso de otros métodos, generalmente de ML, para encontrar el subconjunto de atributos óptimos. Tienen un coste computacional mayor, pero suelen ofrecer mejores resultados que los anteriores.

Aunque existen técnicas como las anteriormente comentadas para automatizar la tarea de selección, no hay que olvidar la importancia del conocimiento sobre el dominio por parte de la persona que realiza el estudio. Es por eso que en primer lugar aprovecharemos el conocimiento adquirido sobre el *dataset* durante la fase de exploración para acometer las primeras selecciones.

Los campos *appName* han sido útiles durante la fase de creación de características, pero examinándolos detalladamente observamos que ciertos valores se comportan como indicadores directos de ataques. Esto choca con la idea de construir un modelo generalista que iden-

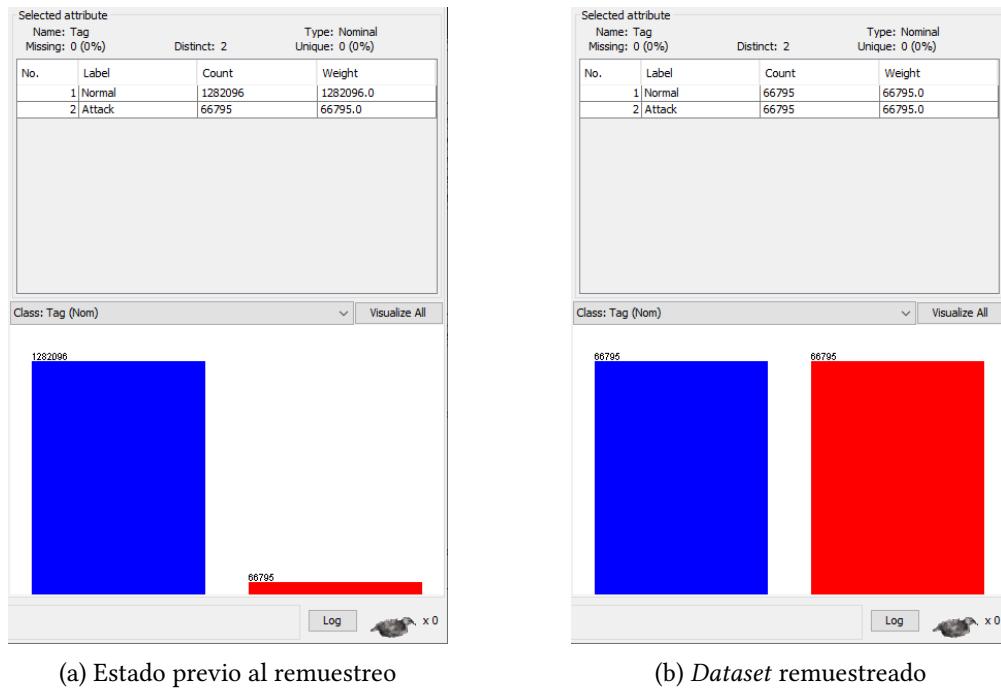


Figura 5.8: Cambios tras realizar un remuestreo 1:1

tifique las anomalías según patrones de tráfico (número de conexiones, tamaño de paquetes, duración) sin tener tan en cuenta el puerto o el servicio a donde va dirigido. El uso de estos atributos *appName* dará lugar a posibles sobreentrenamientos en los modelos. Es por ello que, con la intención de construir un modelo preciso pero generalista, decimos prescindir de ellos.

Para los siguientes pasos [45, cap. 11, sec. 8] haremos uso de la herramienta *Weka* de manera personalizada para cada tipo de modelo.

5.3.1 Regresión logística

Debemos recordar que los modelos de regresión logística son muy susceptibles a la presencia de datos correlacionados. Si hay dos o más variables altamente correlacionadas, podemos tener problemas de sobreestimación o de convergencia del modelo, por lo que en primer lugar procederemos a eliminarlos haciendo uso de un diagrama de correlación.

CAPÍTULO 5. PREPARAR LOS DATOS

appName	Tag	
BitTorrent	Attack	58
	Normal	10136
DNS	Attack	73
	Normal	200766
FTP	Attack	216
	Normal	9399
HTTPImageTransfer	Attack	64
	Normal	453346
HTTPWeb	Attack	40351
	Normal	478625
IMAP	Attack	138
	Normal	6858
IPSec	Attack	29
	Normal	65
MS-SQL	Attack	60
	Normal	13
MSTerminalServices	Attack	66
	Normal	31
MiscApp	Attack	857
	Normal	365

Figura 5.9: Muestras de número de ataques agrupados por appName y Tag

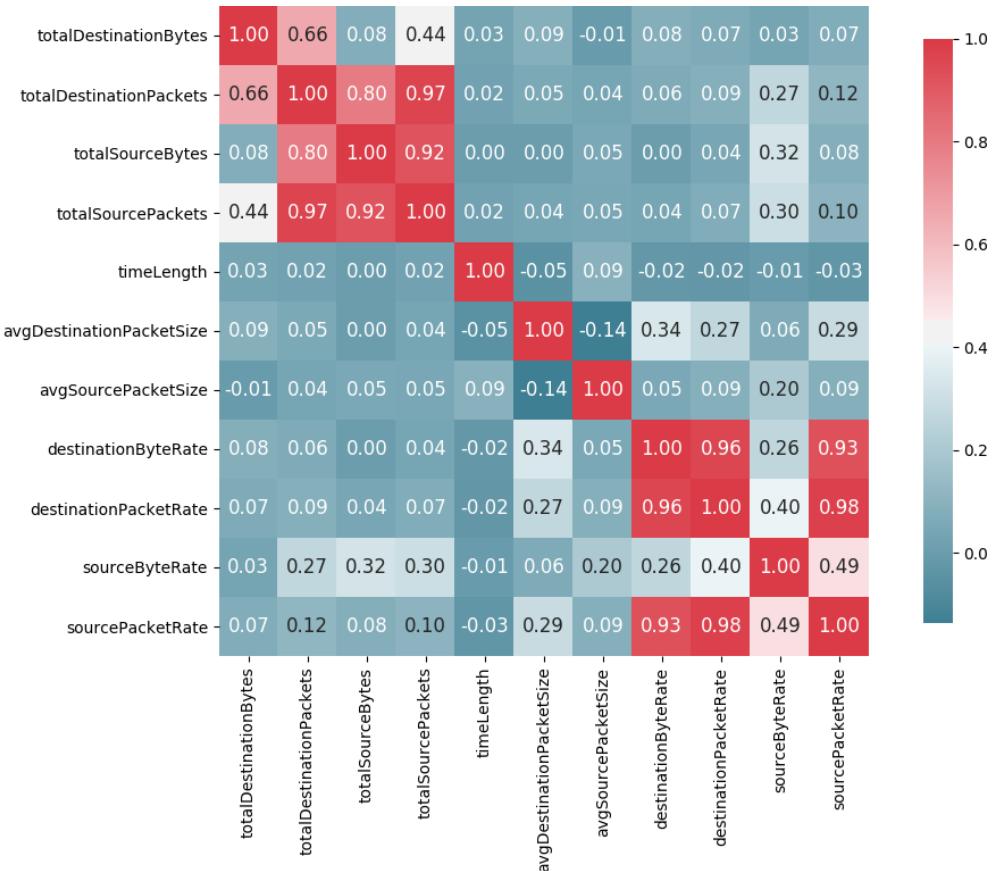


Figura 5.10: Diagrama de correlación de los principales atributos numéricos

En aquellas duplas de valores que presenten una gran relación, como puede ser $totalSourcePackets - totalDestinationPackets$, será necesario eliminar uno de ellos. En la gráfica observamos que la correlación no se da entre los datos originales y los derivados, sino que solo entre ellos mismos. El valor de corte lo asignamos a 0.8, por lo tanto, eliminamos:

- totalSourceBytes
- totalSourcePackets
- destinationPacketRat
- totalSourceBytesDiffMedianScal
- totalSourcePacketsDiffMedianScal
- destinationPacketRatDiffMedianScal

Eliminamos también los atributos referentes a las diferencias sobre la mediana que se basaron en ellos.

Ahora estamos en disposición de aplicar un método de selección de características y de los muchos que nos ofrece Weka nos decidimos por *CfsSubseteval*, que nos permite realizar la selección basándonos en el criterio de máxima relación con la clase objetivo y mínima relación con el resto de atributos. La selección no ha presentado ningún punto de corte claro y ya que hemos eliminado los valores correlacionados, optaremos por una aproximación poco agresiva y reduciremos los atributos a la mitad [A.4.1](#).

5.3.2 Random Forest

Con el fin de poder comparar los algoritmos no solo en precisión sino también en tiempo de entrenamiento y ejecución, vemos conveniente mantener en la medida de lo posible los *datasets* que usaremos para la creación de los modelos lo más parejos posibles. Es por ello que, aunque no es necesario ya que *Random Forest* es un método rápido y preciso con *datasets* de alta dimensionalidad, sí aplicaremos un método de selección.

El método escogido en esta ocasión es el de ganancia de información. Es de nuevo un método basado en filtros y que consiste en calcular la entropía [46] de cada atributo respecto a la clase objetivo.

0.4666739	4 totalSourcePackets
0.4666225	10 sourcePacketRate
0.3325416	62 destinationResume_mainServer
0.1294923	36 sourceTCPFlag_S
0.1247091	7 timeLength

Figura 5.11: Detalle de la salida del método de selección de atributos para el *dataset* de *Random Forest*

Identificamos un punto de corte entre 0.4666225 y 0.3325416 en la salida del método. De esta manera nos quedaríamos solo con 23 atributos. Ya que el método de *Random Forest* se comporta bien tanto computacionalmente como en precisión con conjuntos de alta dimensionalidad, para poder realizar una comparación en igualdad de condiciones con el resto de métodos seleccionaremos los primeros 28 atributos [A.4.2](#) como en el anterior caso.

5.3.3 SVM

Como el anterior, SVM es un modelo que se comporta bien en términos de precisión con problemas de alta dimensionalidad, pero al contrario que *Random Forest* su costo computacional es elevado. Por lo cual en esta ocasión la selección no es opcional. Mantendremos el criterio de 28 atributos por motivos comparativos entre los tres modelos y volveremos a usar ganancia de información como selector [A.4.3](#).

Capítulo 6

Modelar

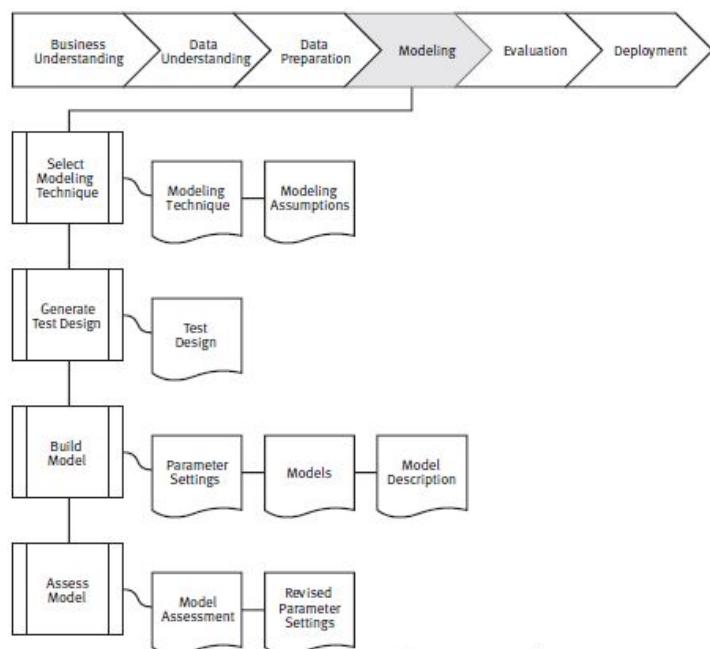


Figura 6.1: Fase CRISP-DM: Modelar.

6.1 Asunciones sobre los modelos

Consiste en la selección 3.2 del algoritmo a modelar y concretar las características del mismo. En este último punto se engloban decisiones como el tipo de atributos posibles en el modelo, necesidad de escalado, tipo de clase objetivo... Estas decisiones han sido expuestas en las secciones anteriores y las mostramos a continuación en una tabla a modo de resumen.

Modelo	Clase objetivo	Escalado	Atributos	Limpieza de valores atípicos	Número de muestras
Regresión logística	Categórica	No	Númericos	Sí	133590
Random forest	Categórica	No	Númericos*	Sí*	133590
SVM	Categórica	Sí	Númericos	Sí*	133590

*No necesario obligatoriamente en el modelo

Tabla 6.1: Resumen del estado del *dataset*

6.2 Validación

Durante esta fase [47, cap. 5] [39, cap. 17] trazaremos la metodología de validación y definiremos la métrica de rendimiento usada para validar el modelo.

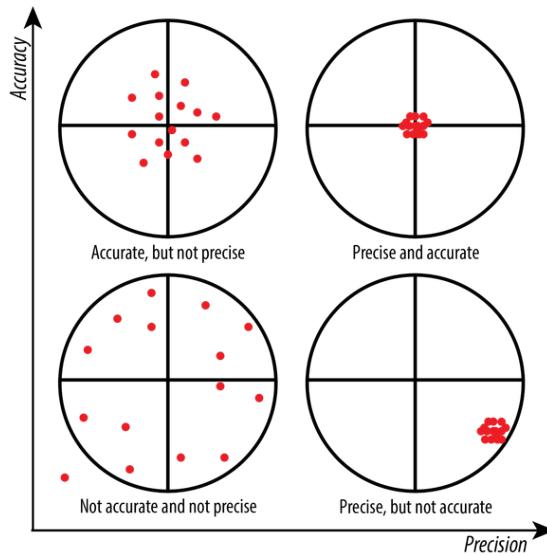
6.2.1 Métricas de rendimiento

La matriz de confusión [18, cap. 6] es una de las formas por la cuales un modelo de clasificación muestra su distribución de errores.

		Real	
		Tag	Normal
Estimado	Normal	TP	FP
	Attack	FN	TN

Tabla 6.2: Matriz de confusión

- TP (*True Positive*): Elemento válido clasificado como válido
- TN (*True Negative*): Elemento inválido clasificado como inválido
- FP (*False Positive*): Elemento válido clasificado como inválido
- FN (*False Negative*): Elemento inválido clasificado como válido

Figura 6.2: Precisión (*Accuracy*) vs Exactitud (*Precision*)

A partir de estos valores podemos calcular diferentes métricas [48]:

- Precisión (*Accuracy*): se refiere a la dispersión del conjunto de valores obtenidos a partir de mediciones repetidas de una magnitud. Cuanto menor es la dispersión mayor la precisión.

$$\text{Precisión} = \frac{TP + TN}{TP + FP + TN + FN}$$

- Exactitud (*Precision*): se refiere a lo cerca que está el resultado de una medición del valor verdadero. La exactitud está relacionada con el sesgo de una estimación.

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

- Sensibilidad (*Recall*): se refiere a la fracción de verdaderos positivos.

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

- Especificidad: se refiere a la fracción de verdaderos negativos.

$$\text{Sensibilidad} = \frac{TN}{TN + FP}$$

- F-score: se refiere a la proporción entre la exactitud y la sensibilidad. La importancia relativa se controla a través del parámetro β . Si β es 1 entonces nos encontramos ante

la media harmónica de F. Es este su valor de uso más común.

$$Fscore = \frac{(1 + \beta^2)Exactitud * Sensibilidad}{\beta^2 * Exactitud + Sensibilidad}$$

El uso de una métrica en lugar de otra vendrá determinado por el proyecto y el coste (tangible o intangible) asociado al error de clasificación del modelo. En nuestro caso en que sí es asumible un falso positivo, pero el riesgo derivado frente a no detectar una anomalía es muy grande, nuestra mejor métrica será la exactitud (*Precision*).

6.2.2 Validación cruzada

La implementación más sencilla para realizar el entrenamiento y validación de un modelo es la partición aleatoria del conjunto de datos en dos conjuntos, entrenamiento y validación. Esta partición se suele realizar en una proporción 75% - 25% y nos permite entrenar nuestro modelo y evitar la sobreestimación producida si se usase todo el conjunto para ambas acciones.

El mayor problema que tiene esta aproximación es que el resultado es dependiente del modo en el que se ha hecho el reparto entre los dos grupos. Esta problemática puede ser muy significativa cuando se nos presentan *dataset* muy desbalanceados, característica que se puede amplificar como resultado de la partición. Tampoco podemos olvidar que esta repartición reduce el número de datos disponibles para realizar el entrenamiento, hecho que se acentúa en *datasets* pequeños.

Con el fin de solventar esta problemática se hace uso de la técnica denominada validación cruzada (*cross-validation*)

Esta técnica consiste en dividir el conjunto de datos en k subconjuntos disjuntos de similar tamaño, véase figura 6.3, y reducir de este modo la dependencia al reparto de datos que presentaba la anterior técnica. Los conjuntos resultantes se utilizarán para realizar las tareas de entrenamiento y validación. La otra ventaja sobre el anterior método es el uso que se hace del conjunto de datos, en esta ocasión siendo $k=10$ usamos el 90% de los datos para entrenar el modelo

La mayor desventaja que supone este modelo es el coste computacional debido a la división del *dataset* y a la realización de los procesos de entrenamiento y validación tantas veces como el número de los subconjuntos k .

Existe una técnica complementaria a la validación cruzada que mejora el proceso de creación de los subconjuntos denominada estratificación que consiste en asegurar que la proporción entre las clases objetivo se mantiene igual que las del conjunto inicial en cada uno de los subconjuntos creados.

Conviene recordar llegado a este punto que, aunque el *dataset* con el que trabajamos simula diferentes escenarios de ataque 4.1 posee una etiqueta única. Si queremos construir un

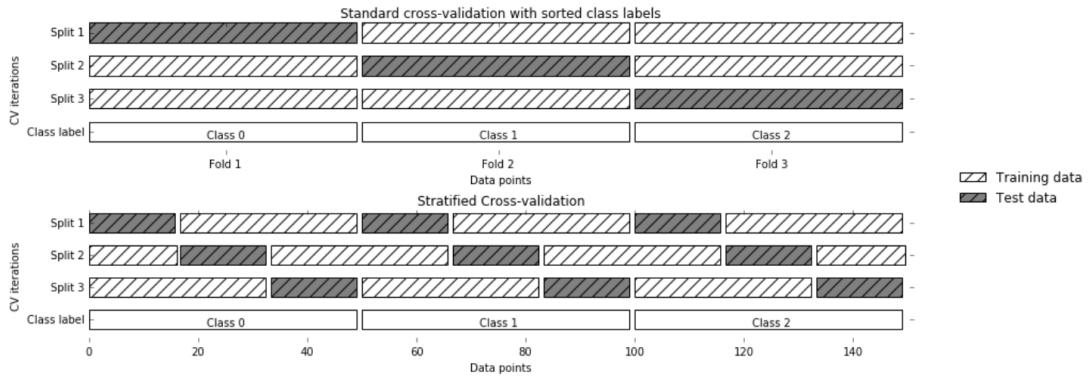


Figura 6.3: Validación cruzada y estratificación

modelo acorde con estos escenarios es necesario garantizar que, durante la creación de los subconjuntos por parte de la validación cruzada, la proporción de tipo de ataque, fig. 6.4 sea la misma a través de todos los subconjuntos. Como ya hemos visto de esta tarea se encarga la estratificación, pero para que funcione del modo que queremos deberemos indicarle que la partición se haga sobre el atributo *typeAttack* 4.2 en lugar de la clase objetivo *Tag*. Esta optimización se puede observar en los diferentes flujos de trabajo de las figuras 6.5, 6.9, 6.14.

6.3 Hiperparametrización

Es importante no confundir este término con parametrización a secas. En el caso de ML parametrización se refiere a los parámetros internos del modelo y que usa para obtener los resultados. Ejemplo de ello son los coeficientes de una regresión logística o los pesos de una red de neuronas.

Cuando hablamos de hiperparametrización nos referimos a aquellos parámetros con los que se configura el modelo, como por ejemplo el parámetro C del algoritmo SVM.

Hiperparametrizar consiste en ajustar estos últimos parámetros para optimizar la precisión final del modelo. Este proceso se suele realizar acompañado de validación cruzada para minimizar la dependencia de los datos sobre el resultado de la parametrización.

Debido al tamaño del *dataset*, que incluso con el inframuestreo ronda los 130 mil datos, y a las limitaciones en cuanto a potencia de la máquina en la que se realiza el estudio, la fase de hiperparametrización se realizará con sólo el 10% de los datos.

6.3.1 Regresión logística

Debido a que la implementación escogida en Weka tiene incluida la validación cruzada con hiperparametrización, no será necesario en esta ocasión definir los valores de búsqueda

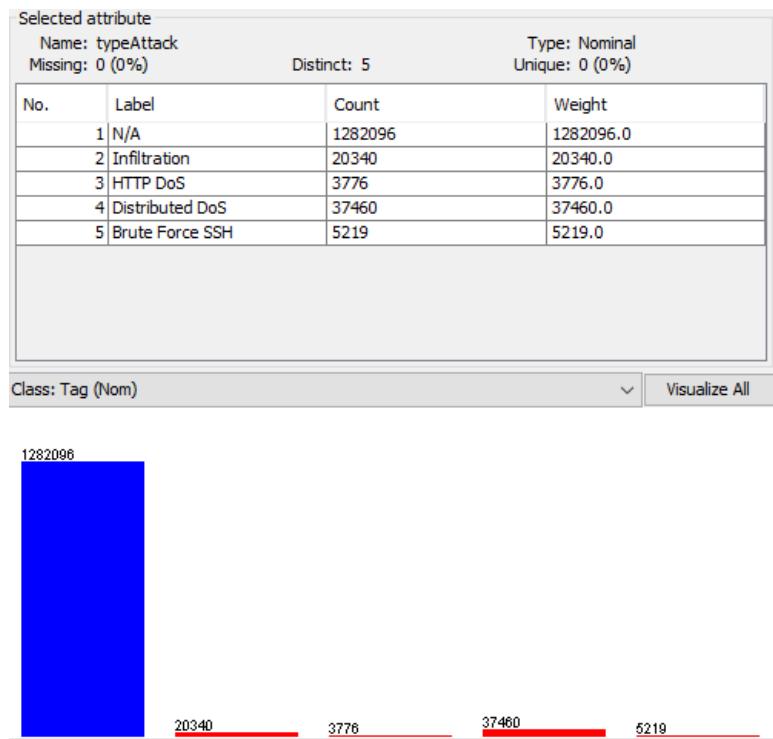


Figura 6.4: Distribución de valores del atributo *typeAttack*

de estos parámetros [49].

El parámetro que optimiza la validación cruzada interna es el número de iteraciones que realiza el modelo para minimizar el error de clasificación del conjunto de entrenamiento.

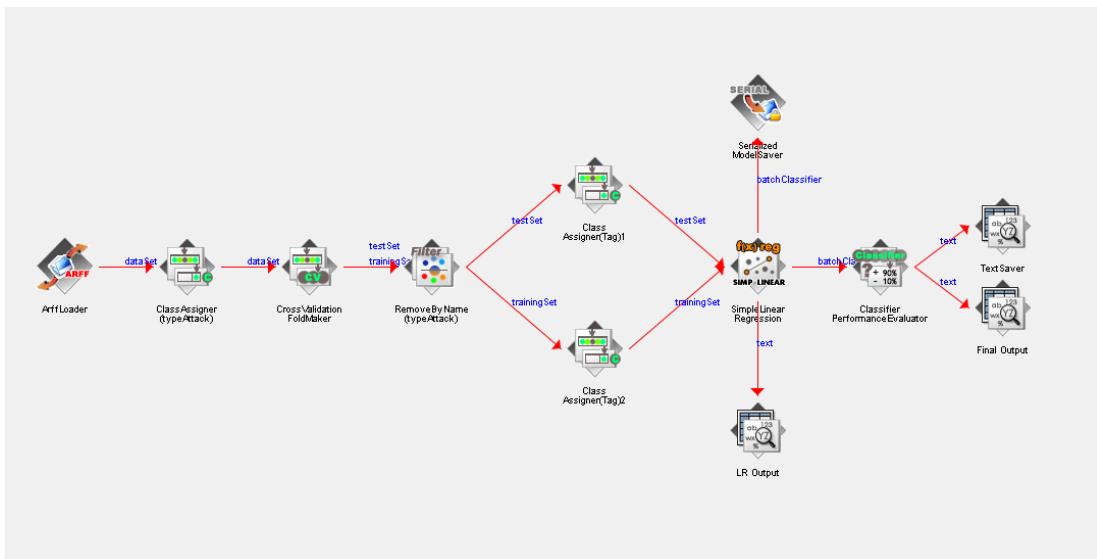


Figura 6.5: Flujo de modelado de Regresión Logística

```
Time taken to build model: 25.24 seconds

==== Stratified cross-validation ====
==== Summary ====

Correctly Classified Instances      129504           96.9414 %
Incorrectly Classified Instances    4086            3.0586 %
Kappa statistic                   0.9388
Mean absolute error               0.0565
Root mean squared error          0.1598
Relative absolute error          11.3022 %
Root relative squared error     31.9556 %
Total Number of Instances        133590

==== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall   F-Measure  Class
          0,955    0,017    0,983     0,955    0,969     Normal
          0,983    0,045    0,957     0,983    0,970     Attack
Wt Avg.    0,969    0,031    0,970     0,969    0,969

== Confusion Matrix ==

      a      b  <-- classified as
63815 2980 |      a = Normal
1106 65689 |      b = Attack
```

Figura 6.6: Resultado final del modelo basado en regresión logística

El resultado en términos de exactitud ha sido bueno, pero analizando los atributos seleccionados y sus coeficientes nos surge la duda de si el modelo se ha visto influido por ruido en las muestras y se ha sobreajustado a él. En la fase de evaluación tras validarla con todo el conjunto de datos (*dataset* sin remuestrear) esperamos encontrar la respuesta.

```
Class Attack :
-0.71 +
[destinationByteRate] * 0      +
[protocolName_tcp_ip] * 0.56 +
[protocolName_udp_ip] * -0.93 +
[sourceTCPFlag_S] * 1.11 +
[destinationPortResume_registered] * 0.35 +
[sourceResume_external] * 0.38 +
[destinationResume_external] * -3.14 +
[destinationResume_internal] * 0.41
```

Figura 6.7: Coeficientes de la función de regresión

6.3.2 Random Forest

El principal parámetro de configuración en *Random Forest* es el número de atributos a elegir en cada una de las divisiones del árbol. En la implementación que usa Weka, cuando ese valor es 0 se define como $\text{int}(\log_2(\text{numerodeatributos}) + 1)$. El otro parámetro que hemos optimizado es el número de árboles creados.

```
==== Classifier model (full training set) ===

Cross-validated Parameter selection.
Classifier: weka.classifiers.trees.RandomForest
Cross-validation Parameter: '-I' ranged from 100.0 to 1300.0 with
    5.0 steps
Cross-validation Parameter: '-K' ranged from 0.0 to 16.0 with 5.0
    steps
Classifier Options: -I 700 -K 12 -P 100 -attribute-importance
    -num-slots 8 -M 1.0 -V 0.001 -S 1
```

Figura 6.8: Hiperparámetros candidatos y finales para *Random Forest*

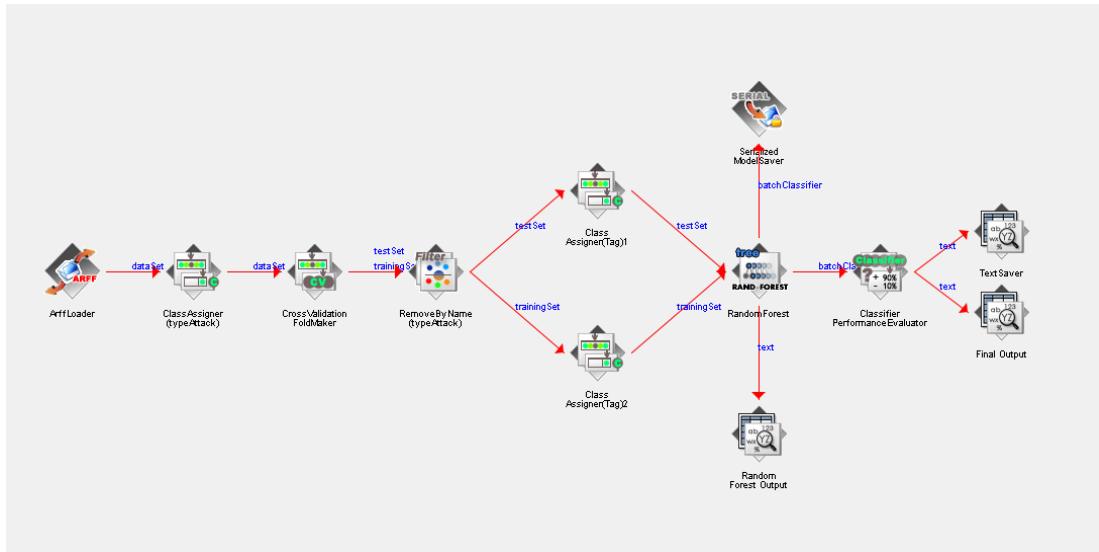


Figura 6.9: Flujo de modelado de *Random Forest*

Una particularidad del algoritmo es que puede funcionar como un método de selección de atributos ya que a su salida devuelve los atributos clasificados según su peso en el modelo [50] A.6.2.

```

Time taken to build model: 287.52 seconds

==== Stratified cross-validation ====
==== Summary ====

Correctly Classified Instances      133197          99.7058 %
Incorrectly Classified Instances     393           0.2942 %
Kappa statistic                      0.9941
Mean absolute error                  0.0052
Root mean squared error              0.0498
Relative absolute error              1.0358 %
Root relative squared error         9.9612 %
Total Number of Instances           133590

==== Detailed Accuracy By Class ====

          TP Rate   FP Rate   Precision   Recall   F-Measure   Class
          0,997     0,003     0,997     0,997     0,997     Normal
          0,997     0,003     0,997     0,997     0,997     Attack
Wt Avg.    0,997     0,003     0,997     0,997     0,997

==== Confusion Matrix ====

      a      b  <-- classified as
66626  169 |      a = Normal
  224 66571 |      b = Attack

```

Figura 6.10: Resultado final del modelo basado en *Random Forest*

6.3.3 SVM

En este algoritmo tenemos tres frentes abiertos a la hora de configurarlo:

- C : es el parámetro que ajusta la complejidad y flexibilidad del hiperplano que separa las dos clases. Con su valor a 0 se comporta como SVM *Hard-Margin*, haciendo difícil que el modelo se ajuste a escenarios complejos, mientras que al aumentar su valor aumenta la flexibilidad y por tanto más puntos son tenidos en cuenta como vectores de soporte.
- *Kernel*: permite parametrizar el uso de diferentes *kernels* que permiten al hiperplano que separan las clases adaptar formas curvas o más complejas.
- Parámetros del *kernel*: en el caso de hacer uso de un *kernel* diferente al polinómico nos podremos encontrar con parámetros adicionales. Un ejemplo sería el uso de un *kernel* gaussiano, que nos permitiría configurar el parámetro Γ . Cuando Γ es muy pequeño el modelo es demasiado limitado y no puede capturar la complejidad de los datos; en el

caso contrario nos encontramos con la situación en el que el modelo es excesivamente concreto produciendo situaciones de sobreajuste.

Las pruebas para este modelo fueron complicadas de realizar ya que Weka no soporta de forma nativa parametrizar parámetros anidados, como ocurre en el caso de Γ . Hubo que realizar pruebas sin hacer validación cruzada haciendo uso de la herramienta *Experimenter* de Weka. Pudimos comprobar como el *kernel* gaussiano se comportaba mejor que el polinómico en términos de tiempo de ejecución manteniendo exactitudes similares.

```
Analysing: IR_precision
Dataset          (1) functions | (2) functi (3) functi (4) functi
-----
'flows_normalized (10)    0.9930 |   0.9927     0.9927   0.9937
-----
(v/ /*) | (0/1/0) (0/1/0) (0/1/0)
Key:
(1) functions.SMO '-C 100.0 functions.supportVector.PolyKernel
(2) functions.SMO '-C 500.0 functions.supportVector.PolyKernel
(3) functions.SMO '-C 100.0 functions.supportVector.RBFKernel
(4) functions.SMO '-C 500.0 functions.supportVector.RBFKernel
```

Figura 6.11: Comparativa de *kernels* respecto a la exactitud

```
Analysing: Elapsed_Time_training
Dataset          (1) function | (2) functi (3) funct (4) funct
-----
'flows_normalized (10)    53.37 |   249.33 v   14.42 *   25.73
-----
(v/ /*) | (1/0/0) (0/0/1) (0/1/0)
Key:
(1) functions.SMO '-C 100.0 functions.supportVector.PolyKernel
(2) functions.SMO '-C 500.0 functions.supportVector.PolyKernel
(3) functions.SMO '-C 100.0 functions.supportVector.RBFKernel
(4) functions.SMO '-C 500.0 functions.supportVector.RBFKernel
```

Figura 6.12: Comparativa de *kernels* respecto al tiempo

```
==== Classifier model (full training set) ===

Cross-validated Parameter selection.
Classifier: weka.classifiers.functions.SMO
Cross-validation Parameter: '-C' ranged from 5000.0 to 10000.0 with
5.0 steps
Classifier Options: -C 5000 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K
"weka.classifiers.functions.supportVector.RBFKernel -C 0 -G
10.0"
-calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1
-num-decimal-places 4" -do-not-check-capabilities
```

Figura 6.13: Hiperparámetros candidatos y finales para SVM

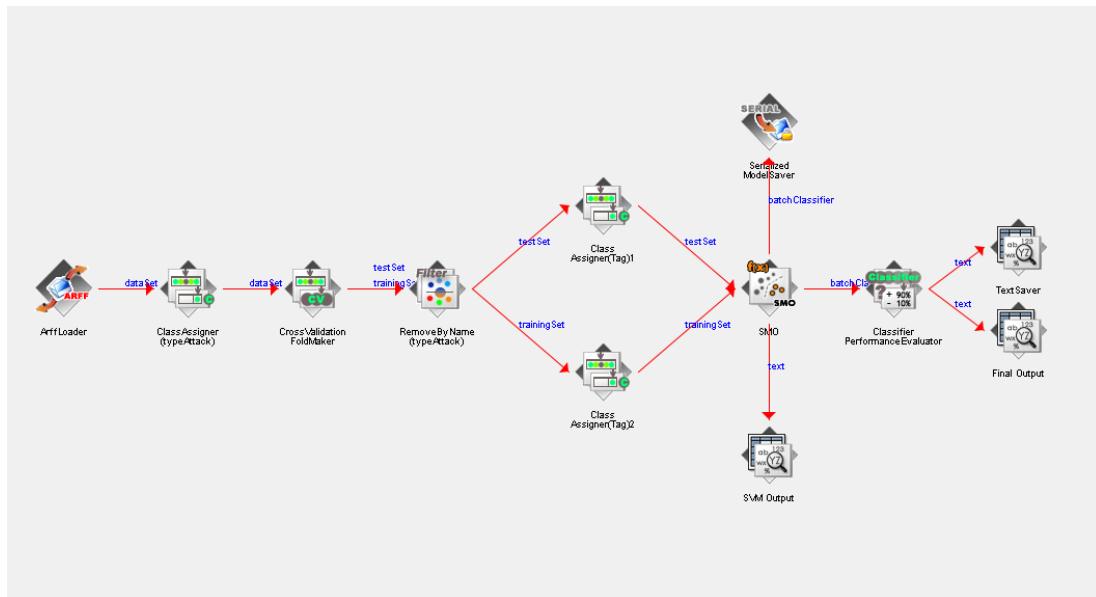


Figura 6.14: Flujo de modelado de SVM

En esta ocasión no pudimos probar los parámetros sobre el conjunto total de los datos debido a limitaciones de la máquina o de la herramienta Weka. Con valores altos de C se producía un problema con el *heap* de la JVM sobre la que corre el *framework* que no fuimos capaces de solucionar. No se valoró cambiar el algoritmo por otro para evitar perder todo el trabajo previo realizado.

```
Time taken to build model: 628.2 seconds

==== Stratified cross-validation ====
==== Summary ====

Correctly Classified Instances      13247          99.1616 %
Incorrectly Classified Instances     112           0.8384 %
Kappa statistic                      0.9832
Mean absolute error                  0.0084
Root mean squared error              0.0916
Relative absolute error              1.6768 %
Root relative squared error         18.3127 %
Total Number of Instances            13359

==== Detailed Accuracy By Class ====

          TP Rate   FP Rate   Precision   Recall   F-Measure   Class
          0,993     0,010     0,990     0,993     0,992     Normal
          0,990     0,007     0,993     0,990     0,992     Attack
Wt Avg.       0,992     0,008     0,992     0,992     0,992

==== Confusion Matrix ====

      a      b    <-- classified as
6634  46 |      a = Normal
 66 6613 |      b = Attack
```

Figura 6.15: Resultado de la validación cruzada del modelo basado en SVM

Capítulo 7

Evaluar

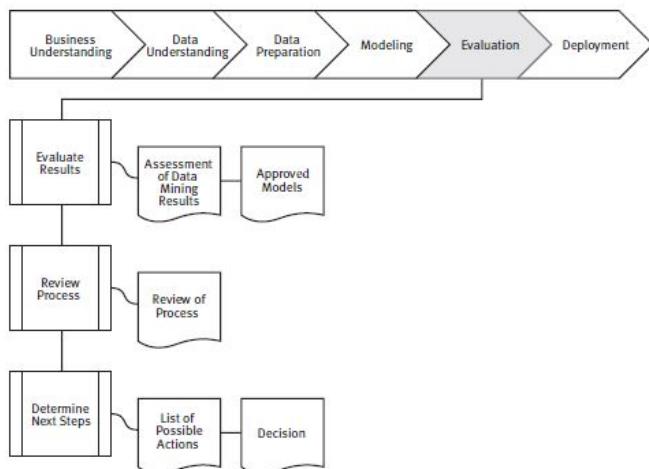


Figura 7.1: Fase CRISP-DM: Evaluar.

ANTES de desgranar las fases, es importante recalcar que el fin último de un proyecto de aprendizaje automático de modelado predictivo, es realizar predicciones lo más precisas posibles y no interpretar los resultados. Por lo tanto, si durante el proceso nos encontramos con que hemos de variar nuestras suposiciones iniciales no hay ningún problema mientras los modelos se mantengan robustos y su funcionamiento sea coherente con el punto de vista del negocio.

Con todos los datos sobre la mesa solo nos queda comparar el rendimiento de las tres opciones estudiadas.

En las figuras 6.6, 6.10 y 6.15 tenemos las métricas finales de cada uno de los algoritmos a las que podemos añadir el tiempo de entrenamiento que han consumido cada uno de ellos.

De entrada, eliminaremos el algoritmo SVM de la selección debido al problema con su ejecución y la imposibilidad de tener un criterio basado en datos reales para tenerlo en cuenta.

De todos modos, su comportamiento en cuanto a tiempos durante todas las fases fue el peor de todos ellos.

```
Regresión logística:  
Time taken to build model: 25.24 seconds  
      TP Rate  FP Rate  Precision  Recall   F-Measure  Class  
      0,955    0,017    0,983     0,955    0,969     Normal  
      0,983    0,045    0,957     0,983    0,970     Attack  
Wt Avg.    0,969    0,031    0,970     0,969    0,969  
== Confusion Matrix ===  
      a      b  <- classified as  
63815 2980 |      a = Normal  
1106 65689 |      b = Attack  
-----  
Random forest:  
Time taken to build model: 287.52 seconds  
      TP Rate  FP Rate  Precision  Recall   F-Measure  Class  
      0,997    0,003    0,997     0,997    0,997     Normal  
      0,997    0,003    0,997     0,997    0,997     Attack  
Wt Avg.    0,997    0,003    0,997     0,997    0,997  
== Confusion Matrix ===  
      a      b  <- classified as  
66626 169 |      a = Normal  
224 66571 |      b = Attack
```

Figura 7.2: Comparativa final entre regresión logística y *Random Forest*

También hemos evaluado ambos algoritmos con la totalidad del *dataset* para comprobar si existía algún problema de sobreajuste y medir el tiempo de ejecución del modelo ya entrenado.

```
Regresión logística:  
Time taken to build model: 24.00 seconds  
          TP Rate  FP Rate  Precision  Recall   F-Measure  Class  
          0,955     0,015    0,999     0,955    0,976    Normal  
          0,985     0,045    0,530     0,985    0,690    Attack  
Wt Avg.    0,956     0,016    0,976     0,956    0,962  
==== Confusion Matrix ====  
      a      b  <- classified as  
1223808  58288 |      a = Normal  
    975    65820 |      b = Attack  
  
-----  
Random forest:  
Time taken to build model: 305.00 seconds  
          TP Rate  FP Rate  Precision  Recall   F-Measure  Class  
          0,998     0,002    1,000     0,998    0,999    Normal  
          0,998     0,002    0,961     0,998    0,979    Attack  
Wt Avg.    0,998     0,002    0,998     0,998    0,998  
==== Confusion Matrix ====  
      a      b  <- classified as  
1279368  2728  |      a = Normal  
    114    66681 |      b = Attack
```

Figura 7.3: Comparativa sobre la validación del *dataset* completo entre regresión logística y *Random Forest*

Como sospechábamos, regresión logística sufre un problema de sobreajuste sobre la etiqueta *Normal*. En la prueba de validación la exactitud respecto a la clase *Attack* cae hasta el 50%.

Con ambas pruebas realizadas y siguiendo el criterio de *Precision* definido durante el apartado 6.2.1 se alza ganador *Random Forest*.

Capítulo 8

Desplegar

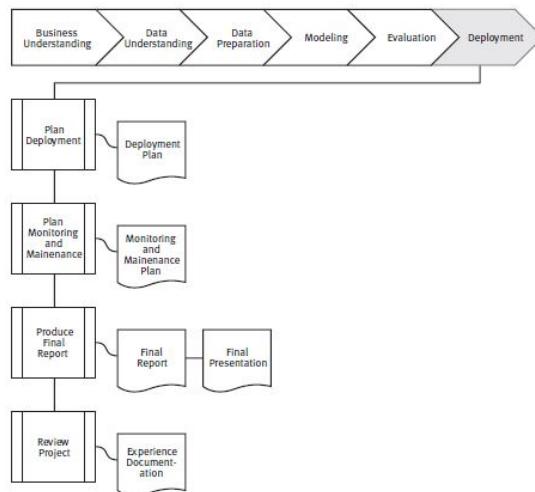


Figura 8.1: Fase CRISP-DM: Desplegar.

DEBIDO a que el proyecto se realizó en diferentes etapas y usando diferentes tecnologías no podemos hacer un despliegue directo del modelo. Esta fase consistirá en implementar el algoritmo *Random Forest* en la plataforma Databricks según los hiperparámetros identificados durante la fase de modelado.

Para el uso de Databricks en su versión no limitada a un sólo *cluster* de una sola máquina, es necesario darse de alta en alguno de los planes que ofrece. Durante el proceso también es necesario escoger la infraestructura que usará como soporte Databricks: Azure o AWS.

El primer paso es crear el *cluster* donde ejecutar el modelo, con este fin lo configuraremos con dos trabajadores y un controlador.

(a) Creación de *clusters* en Databricks

(b) Resumen del *cluster*

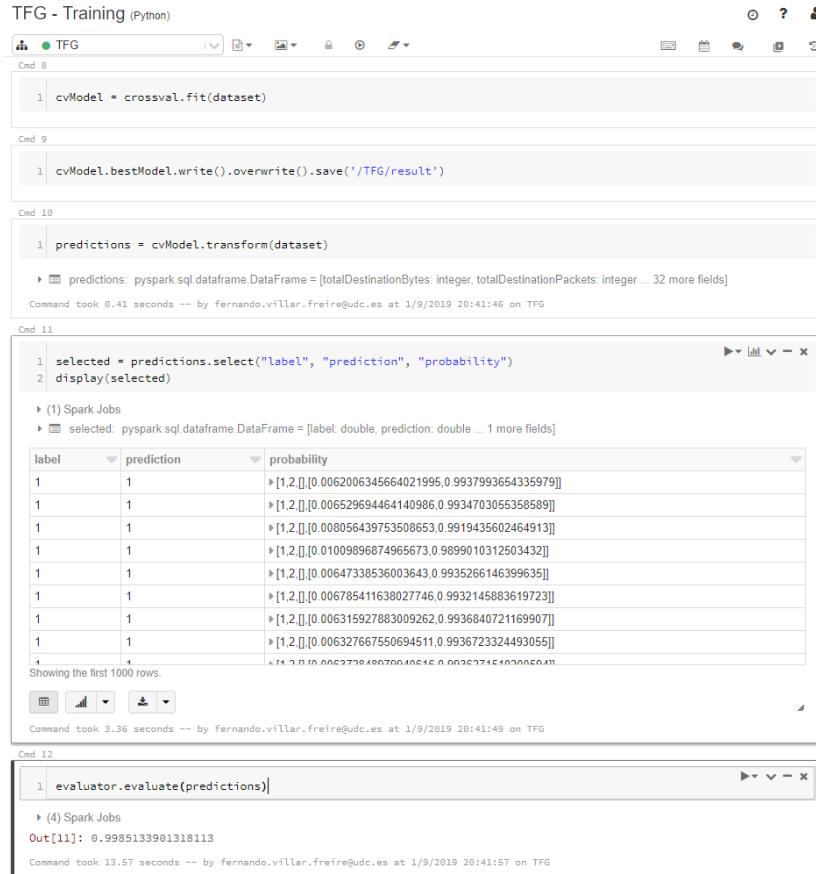
Figura 8.2: *Clusters* en Databrick

Como ya comentamos en el cap. 3.5 Databricks ofrece los *notebooks* como interfaz con la que interactuar con el sistema Spark que nos proporciona. La interfaz de trabajo es similar al formato de *notebook* Jupyter de Python donde se nos permite escribir código y ejecutarlo a continuación.

Paso previo a implementar nuestro modelo será cargar el *dataset* de trabajo a través de la interfaz web que se nos proporciona. El sistema de archivos DBFS[51], junto con el uso de *Dataset* y *Dataframes* [52] que acompaña a Spark, permite que las tareas relacionadas a la distribución de los datos en el *cluster* sean transparentes para el usuario.

La librería MLlib que se encarga de proporcionar implementación distribuida a las funciones de ML proporciona todas las herramientas usadas durante las anteriores etapas en Weka, a excepción de la validación cruzada con estratificación.

Tras realizar el entrenamiento y la validación comprobamos que los resultados son coherentes con los alcanzados en el *framework* Weka.



The screenshot shows a Databricks notebook titled "TFG - Training (Python)". It contains four command cells (Cmd 8, Cmd 9, Cmd 10, Cmd 11) and one command cell (Cmd 12).

- Cmd 8:**

```
1 cvModel = crossval.fit(dataset)
```
- Cmd 9:**

```
1 cvModel.bestModel.write().overwrite().save('/TFG/result')
```
- Cmd 10:**

```
1 predictions = cvModel.transform(dataset)
```

↳ predictions: pyspark.sql.dataframe.DataFrame = [totalDestinationBytes: integer, totalDestinationPackets: integer ... 32 more fields]

Command took 0.41 seconds -- by fernando.villar.freire@udc.es at 1/9/2019 20:41:46 on TFG
- Cmd 11:**

```
1 selected = predictions.select("label", "prediction", "probability")
2 display(selected)
```

↳ (1) Spark Jobs

↳ selected: pyspark.sql.dataframe.DataFrame = [label: double, prediction: double ... 1 more fields]

label	prediction	probability
1	1	[1.2,[0.0062006345664021995,0.9937993654335979]]
1	1	[1.2,[0.006529694464140986,0.9934703055358589]]
1	1	[1.2,[0.008056439753508653,0.9919435602464913]]
1	1	[1.2,[0.01009896874965673,0.9899010312503432]]
1	1	[1.2,[0.00647338536003643,0.9935266146399635]]
1	1	[1.2,[0.006785411638027746,0.9932145883619723]]
1	1	[1.2,[0.0063159278309262,0.9936840721169907]]
1	1	[1.2,[0.006327667550694511,0.9936723324493055]]
1	1	[1.2,[0.006272946070040616,0.99326271510000504]]

Showing the first 1000 rows.
- Cmd 12:**

```
1 evaluator.evaluate(predictions)
```

↳ (4) Spark Jobs

Out[11]: 0.9985133901318113

Command took 13.57 seconds -- by fernando.villar.freire@udc.es at 1/9/2019 20:41:57 on TFG

Figura 8.3: Exactitud del modelo *Random Forest* en Databricks

Con el modelo resultante guardado, nuestro proyecto está listo para pasar a las pruebas de rendimiento.

8.1 Rendimiento

Uno de los objetivos del proyecto es medir el rendimiento de la implementación distribuida de un algoritmo de ML [53].

Para realizar las pruebas debemos dotarnos de un flujo constante de datos por lo que crearemos un *dataset* nuevo mediante la concatenación del original dando como resultado un nuevo *dataset* de 17 millones de muestras.

Como métrica para medir el rendimiento hemos utilizado la que nos proporciona de forma nativa el uso de *structured streaming* [54]: *Processing Rate*. Con ella podemos medir la capacidad de procesamiento que tiene nuestro *cluster* medido en registros por segundo. La otra métrica *Input Rate* nos da una idea de la capacidad de ingestión a la que se puede someter al conjunto.

8.1. Rendimiento

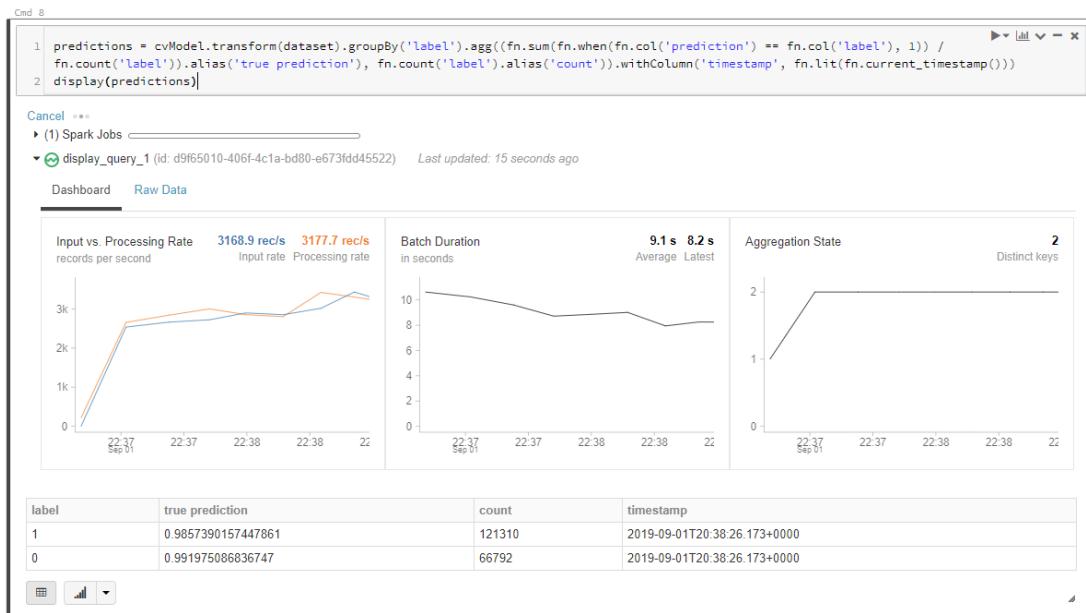


Figura 8.4: Cluster funcionando con dos trabajadores

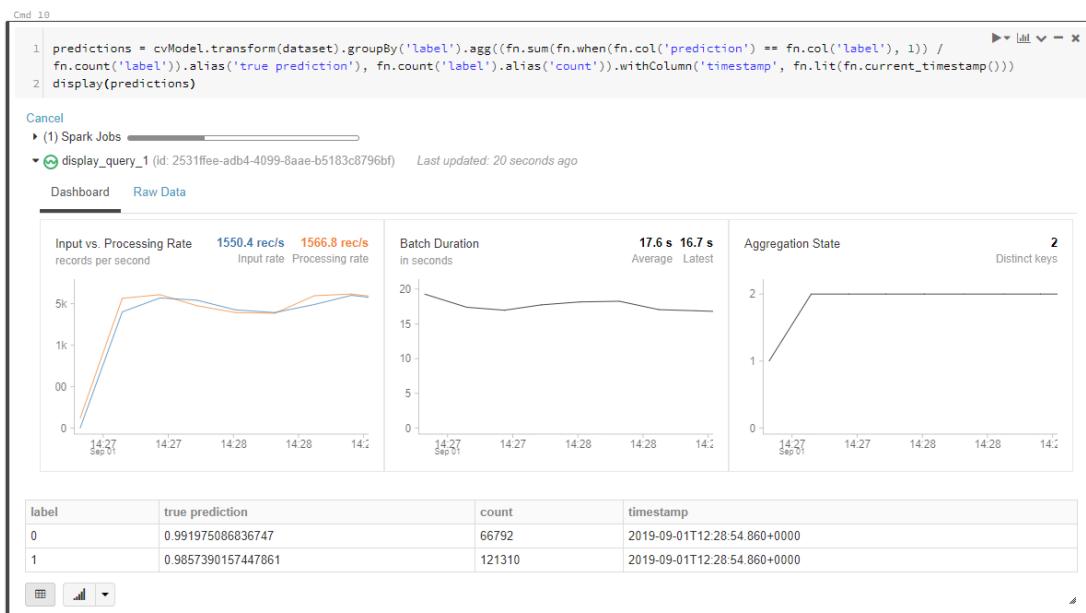


Figura 8.5: Cluster funcionando con cuatro trabajadores

Nos han surgido problemas de configuración a la hora de conseguir que la propia paralelización y distribución de los datos no provocasen un *overhead* que afectase a las mediciones. Esto se ha debido a no usar un ingestador de datos como Kafka y haber usado el propio *cluster* para realizar esa función a través de la funcionalidad de *structured streaming* [55] de Spark.

Structured streaming se introdujo en Spark 2.0 y es una adición a Spark Streaming. La idea principal es tener una tabla que constantemente tenga nuevas entradas agregadas a él. Cada elemento de datos que llega a la secuencia es como una nueva fila que se agrega a la tabla. Debido a que no existe una manera sencilla de asignar a alguno de los nodos trabajadores la función exclusiva de la ingestión, esta tarea terminaba por afectar a todo el conjunto de nodos a través del *overhead* generado por la lectura del *dataset*. A pesar de intentarlo y probar con diferentes combinaciones [56] de parámetros no conseguimos alcanzar resultados razonables, encontrándonos con la paradoja de que un mayor número de nodos trabajadores arrojaba peores resultados en términos de registros leídos y procesados.

También durante las pruebas de rendimiento observamos que el balanceo de carga, sin realizar ninguna configuración adicional, produce que en *clusters* con pocos trabajadores no sea muy equitativo.

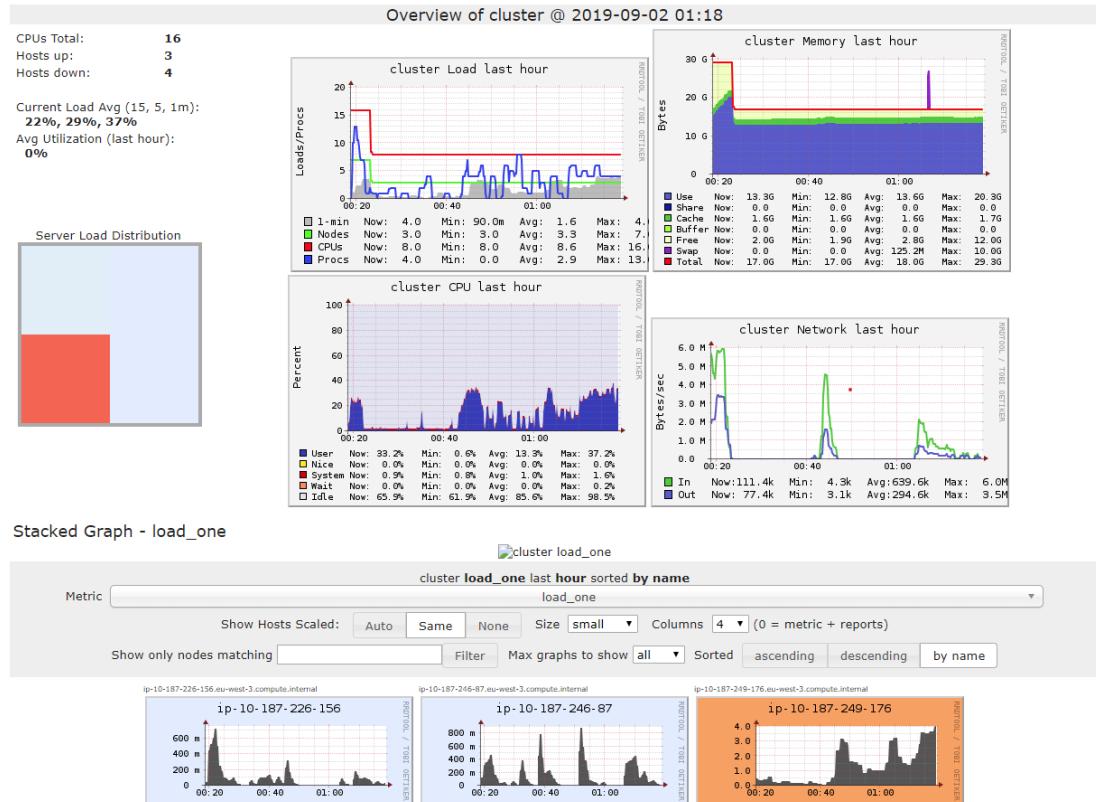


Figura 8.6: Problemas de balanceo en *clusters* de pocos trabajadores

Debido a todo esto no podemos sacar ninguna conclusión sobre el comportamiento de la funcionalidad de detección implementada en un entorno distribuido. Las posibles soluciones que podrían solventar esta problemática las comentaremos en el siguiente capítulo dedicado a las conclusiones y las líneas futuras.

Capítulo 9

Conclusiones y líneas futuras

La realización de este proyecto ha significado para mí una valiosísima oportunidad para aumentar las competencias adquiridas durante el estudio de grado y prepararme para un sector tan en boga y actualidad como es la IA.

Enfrentarme a la tarea de gestionar un proyecto desde la incertidumbre causada por la inexperiencia inicial respecto al tema me permitió abordar la problemática de la gestión de expectativas y solucionar los contratiempos que han aparecido durante el desarrollo llegando a conclusiones tan importantes como que han sido también experiencias provechosas aquellas partes donde no he conseguido los resultados deseados como en las pruebas de rendimiento, pues de ello he aprendido que del error nace también el conocimiento.

Todos los campos vinculados a ML, a pesar de estar alejados de la mención de TI me permitieron explorar técnicas, herramientas y conceptos de rotunda actualidad y muchas de las lecciones asimiladas son debidas a que los conocimientos adquiridos durante la consecución del trabajo me han permitido tener una visión más amplia del propio problema y de las herramientas existentes en el mercado para abordarlo. Ello me hace reflexionar sobre el camino a seguir si tuviese que abordar esta misma tarea desde su inicio.

A lo largo del proyecto conviven dos perfiles diferenciados. Uno orientado a sistemas en las partes enfocadas a los flujos de datos y la arquitectura distribuida, y otro centrado en la ingeniería de datos, el cual supone el núcleo central del proyecto.

La parte de ingeniería de datos la hemos abordado desde un punto de vista integral asumiendo en diferentes etapas del trabajo las perspectivas más frecuentes en la realización de estas tareas: ingeniero de datos y científico de datos.

Como ingeniero de datos hemos asumido las tareas de ingeniería de características y la creación del modelo sobre la plataforma Databricks mediante una implementación programática haciendo uso de Python y Pyspark.

Como científico de datos el uso del *framework* Weka, en su variante GUI, nos posibilitó una aproximación menos técnica a las tareas de modelado.

Weka ha constituido precisamente uno de los aspectos más problemáticos. Se nota que Weka es una de las primeras soluciones en cuanto a herramienta GUI de ML y a pesar de aportar una curva de aprendizaje en su uso básico sencilla, su uso restringido a la GUI se hace limitante y susceptible a errores. Un ejemplo es la herramienta de flujo de conocimiento y el problema con el *executor* de *threads* que imposibilitó realizar *pipelines* complejos sobre alguno de los algoritmos de estudio. A día de hoy hubiera realizado el proyecto enteramente en algunas de las plataformas dedicadas a ello como puede ser la misma Databricks sobre AWS o las pertenecientes a Azure, Google o Tensorflow.

Otro punto a destacar sería que de los cuatro objetivos iniciales (explorar la captura de flujos, implementar y comparar algoritmos de ML, implementar y desplegar una arquitectura distribuida y evaluar el rendimiento de los algoritmos resultantes) tan sólo un apartado del último objetivo no llegó a culminarse debido a que su solución suponía una carga de trabajo adicional que excedía con creces el alcance original del proyecto. Esto fue debido a una estimación errónea en cuanto a la extensión del trabajo en la redacción del anteproyecto por lo cual me vi obligado a tomar la decisión de simplificar la captura e ingestión de datos. Se partió de un *dataset* ya conocido y creado para la ocasión obviando de este modo las fases de captura real de flujos y su etiquetado. Para la ingestión de los datos se desestimaron maneras más complejas como habría sido el uso de Apache Kafka en lugar del propio *structured streaming* de Apache Spark. Todas estas decisiones se tomaron dentro de un contexto de desarrollo ágil en el que opté por la adaptación al cambio y la entrega continua de valor. Cada una de ellas me han llevado a la certeza de que este campo es un escenario mudable y dinámico en el que se debe asumir sin ambages la variabilidad adoptando una postura flexible.

Centrándonos ya en aquellas funcionalidades que por falta de tiempo o extensión del proyecto se quedaron en el tintero destacaría:

- Añadir las transformaciones llevadas a cabo durante la ingeniería de características a la fase distribuida a través de su implementación en la plataforma Databricks y su ejecución en un *pipeline*.
- Uso de Kafka como ingestador de datos para solventar los problemas de medición del rendimiento.
- Implementación de *crossvalidation stratified* en el flujo distribuido de Databricks.
- Creación de un entorno visual propio para mostrar los datos sobre tráfico y su análisis, es decir, implementar el módulo de salida de un NTA.

Apéndices

Apéndice A

Material adicional

A.1 *Sprints* del proyecto

Sprint 0

Historias de usuario	Tareas	Estimaciones
Entender el negocio	Estudio de los fundamentos teóricos asociados a ML	7
	Estudio de los fundamentos teóricos y tecnológicos asociados a los flujos de red	3
	Estudio de los fundamentos teóricos y tecnológicos asociados a los sistemas distribuidos	5
Total		15

Tabla A.1: Tareas del *sprint 0*

Sprint 1

Historias de usuario	Tareas	Estimaciones
Modelo de regresión logística	Estudio detallado del algoritmo regresión logística	1
	Entender los datos	5
	Preparar los datos	4
	Modelar	3
	Evaluuar	2
Total		15

Tabla A.2: Tareas del *sprint 1*

Sprint 2

Historias de usuario	Tareas	Estimaciones
Modelo de <i>random forest</i>	Estudio detallado del algoritmo <i>random forest</i>	5
	Preparar los datos	4
	Modelar	4
	Evaluar	2
	Total	15

Tabla A.3: Tareas del *sprint 2*

Sprint 3

Historias de usuario	Tareas	Estimaciones
Modelo de SVM	Estudio detallado del algoritmo SVM	5
	Preparar los datos	4
	Modelar	4
	Evaluar	2
	Total	15

Tabla A.4: Tareas del *sprint 3*

Sprint 4

Historias de usuario	Tareas	Estimaciones
Despliegue	Configuración del sistema distribuido Databricks	10
	Pruebas de rendimiento	5
Total		15

Tabla A.5: Tareas del *sprint 4*

Sprint 5

Historias de usuario	Tareas	Estimaciones
Redacción de la memoria	Cap. Resumen e introducción	2
	Cap. Planificación del proyecto	0,5
	Cap. Metodología	0,5
	Cap. Entender el negocio	3
	Cap. Entender los datos	1
	Cap. Preparar los datos	2
	Cap. Modelar	2
	Cap. Evaluar	1
	Cap. Desplegar	2
	Cap. Líneas futuras y conclusiones	1
Total		15

Tabla A.6: Tareas del *sprint 5***A.2 Descripción de los atributos iniciales del dataset**

```
In[]:print(all_flows.info(verbose=True))
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1416201 entries, 0 to 1416200
Data columns (total 22 columns):
Tag                           1416201 non-null object
appName                       1416201 non-null object
date                          1416201 non-null object
destination                   1416201 non-null object
destinationPayloadAsBase64    620061 non-null object
destinationPayloadAsUTF       620008 non-null object
destinationPort                1416201 non-null object
destinationTCPFlagsDescription 1366947 non-null object
direction                     1416201 non-null object
protocolName                  1416201 non-null object
source                        1416201 non-null object
sourcePayloadAsBase64          698046 non-null object
sourcePayloadAsUTF            784626 non-null object
sourcePort                     1416201 non-null object
sourceTCPFlagsDescription     1413843 non-null object
startTime                     1416201 non-null datetime64[ns]
stopDateTime                  1416201 non-null datetime64[ns]
totalDestinationBytes         1416201 non-null int64
```

```
totalDestinationPackets      1416201 non-null int64
totalSourceBytes             1416201 non-null int64
totalSourcePackets           1416201 non-null int64
typeAttack                   1416201 non-null object
dtypes: datetime64[ns](2), int64(4), object(16)
memory usage: 237.7+ MB
```

```
Tag:
count      1416201
unique     2
top        Normal
freq       1349388
Name: Tag, dtype: object
0   Normal
1   Normal
2   Normal
3   Normal
4   Normal
5   Normal
6   Normal
7   Normal
8   Normal
9   Normal
Name: Tag, dtype: object
['Normal' 'Attack']
```

```
appName:
count      1416201
unique     107
top        HTTPWeb
freq       518998
Name: appName, dtype: object
0        Unknown_UDP
1        Unknown_UDP
2        HTTPWeb
3        HTTPWeb
4        HTTPWeb
5        HTTPWeb
6    HTTPImageTransfer
7    HTTPImageTransfer
8        SecureWeb
9        SecureWeb
Name: appName, dtype: object
['Unknown_UDP' 'HTTPWeb' 'HTTPImageTransfer' 'SecureWeb' 'IMAP'
 'NetBIOS-IP' 'DNS' 'POP' 'WindowsFileSharing' 'SMTP' 'FTP'
 'MiscApplication' 'WebMediaDocuments' 'ICMP' 'IGMP' 'NTP']
```

APÉNDICE A. MATERIAL ADICIONAL

```
'WebMediaAudio'  
'SSDP' 'DNS-Port' 'IRC' 'TimeServer' 'Unknown_TCP' 'Gnutella'  
 'AOL-ICQ'  
'NETBEUI' 'Tacacs' 'MDQS' 'MSTerminalServices' 'MSMQ' 'POP-port'  
'Common-P2P-Port' 'Anet' 'Ingres' 'VNC' 'Authentication' 'RPC'  
 'PPTP'  
'H.323' 'SunRPC' 'Yahoo' 'RTSP' 'ManagementServices' 'XWindows'  
 'Telnet'  
'Webmin' 'Oracle' 'Flowgen' 'Misc-Ports' 'LDAP' 'NNTPNews' 'XFER'  
 'NFS'  
'Hosts2-Ns' 'Misc-Mail-Port' 'Filenet' 'Misc-DB' 'Timbuktu' 'IPSec'  
'MS-SQL' 'Hotline' 'NortonAntiVirus' 'SNMP-Ports' 'OpenNap' 'MSN'  
 'rsh'  
'Groove' 'Network-Config-Ports' 'dsp3270' 'MiscApp' 'PCAnywhere'  
'SSL-Shell' 'rexec' 'MicrosoftMediaServer' 'BGP' 'Google' 'SMS'  
 'Real'  
'BitTorrent' 'StreamingAudio' 'rlogin' 'Intellex' 'Squid' 'Citrix'  
'Printer' 'PostgreSQL' 'SSH' 'PeerEnabler' 'WebFileTransfer'  
 'Web-Port'  
'Common-Ports' 'GuptaSQLBase' 'OpenWindows' 'Nessus' 'NortonGhost'  
'Kazaa' 'SIP' 'giop-ssl' 'MSN-Zone' 'SAP' 'iChat' 'SNA' 'IPX'  
'UpdateDaemon' 'MGCP' 'Blubster' 'WebMediaVideo' 'TFTP']
```

```
date:  
count      1416201  
unique        4  
top       20100615  
freq      571698  
Name: date, dtype: object  
0    20100613  
1    20100613  
2    20100613  
3    20100613  
4    20100613  
5    20100613  
6    20100613  
7    20100613  
8    20100613  
9    20100613  
Name: date, dtype: object  
['20100613' '20100614' '20100615' '20100617']
```

```
destination:  
count      1416201  
unique      31688  
top       198.164.30.2
```

```

freq          151954
Name: destination, dtype: object
0      224.0.0.251
1      224.0.0.251
2      192.168.5.122
3      192.168.5.122
4      207.241.148.80
5      207.241.148.80
6      216.49.88.12
7      216.49.88.12
8      63.245.209.72
9      63.245.209.72
Name: destination, dtype: object
['224.0.0.251' '192.168.5.122' '207.241.148.80' ... '64.130.7.181'
 '66.135.204.28' '65.54.81.88']

```

```

destinationPayloadAsBase64:
count                  620061
unique                 455716
top       KiBPSyBTdG1sbCBoZXJ1DQoqIE9LIFN0aWxsIGh1cmUNCg==
freq                  5461
Name: destinationPayloadAsBase64, dtype: object
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
5      NaN
6      NaN
7      NaN
8      NaN
9      NaN
Name: destinationPayloadAsBase64, dtype: object
[nan 'KiBPSyBTdG1sbCBoZXJ1DQoqIE9LIFN0aWxsIGh1cmUNCg
xpYy9zdHlsZV9jc3MvY3NzXzEvaXBix211LmNzcyIgLz4NCjwhW2VuZG1mXQ==' ]

```

```

destinationPayloadAsUTF:
count                  620008
unique                 447031
top       * OK Still here* OK Still here
freq                  5461
Name: destinationPayloadAsUTF, dtype: object
0      NaN
1      NaN
2      NaN
3      NaN

```

APÉNDICE A. MATERIAL ADICIONAL

```
4      NaN
5      NaN
6      NaN
7      NaN
8      NaN
9      NaN
Name: destinationPayloadAsUTF, dtype: object
```

```
destinationPort:
count      1416201
unique     22089
top         80
freq       976330
Name: destinationPort, dtype: object
0      5353
1      5353
2        80
3        80
4        80
5        80
6        80
7        80
8      443
9      443
Name: destinationPort, dtype: object
['5353' '80' '443' ... '55930' '34609' '36928']
```

```
destinationTCPFlagsDescription:
count      1366947
unique      28
top        F,S,P,A
freq       756548
Name: destinationTCPFlagsDescription, dtype: object
0      N/A
1      N/A
2        R
3        R
4      F,A
5      F,A
6      F,A
7      F,A
8      NaN
9      NaN
Name: destinationTCPFlagsDescription, dtype: object
['N/A' 'R' 'F,A' 'nan' 'P,A' 'F,S,P,A' 'S,P,A' 'F,S,R,P,A' 'S,R,P,A'
 'F,R,A'
```

```
'A' 'F,P,A' 'S,A' 'F,R,P,A' 'F,S,A' 'F,S,R,A' 'R,A' 'S,R,A'  
'S,R,A,Illegal18' 'R,P,A' 'F,S,P,A,Illegal18' 'S' 'R,A,Illegal17'  
'R,A,Illegal17,Illegal18' 'R,A,Illegal18' 'S,P,A,Illegal18'  
'F,S,R,P,A,Illegal18' 'F,P,A,Illegal18' 'R,Illegal18']
```

```
direction:  
count      1416201  
unique       4  
top        L2R  
freq      1202862  
Name: direction, dtype: object  
0      L2R  
1      L2R  
2      L2L  
3      L2L  
4      L2R  
5      L2R  
6      L2R  
7      L2R  
8      L2R  
9      L2R  
Name: direction, dtype: object  
['L2R' 'L2L' 'R2L' 'R2R']
```

```
protocolName:  
count      1416201  
unique       5  
top        tcp_ip  
freq      1114265  
Name: protocolName, dtype: object  
0      udp_ip  
1      udp_ip  
2      tcp_ip  
3      tcp_ip  
4      tcp_ip  
5      tcp_ip  
6      tcp_ip  
7      tcp_ip  
8      tcp_ip  
9      tcp_ip  
Name: protocolName, dtype: object  
['udp_ip' 'tcp_ip' 'ip' 'icmp' 'icmp_ip']
```

```
source:  
count      1416201  
unique       2438
```

APÉNDICE A. MATERIAL ADICIONAL

```
top      192.168.5.122
freq          175757
Name: source, dtype: object
0      192.168.5.122
1      192.168.5.122
2      192.168.2.113
3      192.168.2.113
4      192.168.2.113
5      192.168.2.113
6      192.168.2.110
7      192.168.2.110
8      192.168.2.113
9      192.168.2.113
Name: source, dtype: object
['192.168.5.122' '192.168.2.113' '192.168.2.110' ...
 '216.114.79.153'
 '58.211.72.43' '194.72.238.61']
```

```
sourcePayloadAsBase64:
count          698046
unique         362574
top      R0VUIC8gSFRUUC8xLjENCkhvc3Q6IDE5Mi4xNjguNS4xMj...
freq          43475
Name: sourcePayloadAsBase64, dtype: object
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
5      NaN
6      NaN
7      NaN
8      NaN
9      NaN
Name: sourcePayloadAsBase64, dtype: object
```

```
sourcePayloadAsUTF:
count      784626
unique     239691
top        nan
freq      171380
Name: sourcePayloadAsUTF, dtype: object
0      NaN
1      NaN
2      NaN
3      NaN
```

```
4      NaN
5      NaN
6      NaN
7      NaN
8      NaN
9      NaN
Name: sourcePayloadAsUTF, dtype: object
```

```
sourcePort:
count      1416201
unique     63905
top        58040
freq       59572
Name: sourcePort, dtype: object
0      5353
1      5353
2      4191
3      4191
4      4192
5      4192
6      1864
7      1864
8      4186
9      4186
Name: sourcePort, dtype: object
['5353' '4191' '4192' ... '32389' '43659' '34384']
```

```
sourceTCPFlagsDescription:
count      1413843
unique      24
top        F,S,P,A
freq       771549
Name: sourceTCPFlagsDescription, dtype: object
0      N/A
1      N/A
2      F,A
3      F,A
4      F,A
5      F,A
6      F,A
7      F,A
8      F,A
9      F,A
Name: sourceTCPFlagsDescription, dtype: object
['N/A' 'F,A' 'A' 'F,S,P,A' 'S,P,A' 'F,S,R,P,A' 'S' 'S,R,P,A' 'F,S,A'
 'F,P,A' 'nan' 'P,A' 'S,A' 'R,A' 'F,R,A' 'R' 'S,R' 'R,P,A' 'F,R,P,A'
```

APÉNDICE A. MATERIAL ADICIONAL

```
'F,P,U'  
'S,R,Illegal17,Illegal18' 'F,S,R,P,U' 'F,S,P,U' 'F,S,R,A' 'S,R,A' ]
```

```
startDateTime:  
count           1416201  
unique          200146  
top      2010-06-13 16:42:25  
freq            9742  
first     2010-06-12 23:57:24  
last      2010-06-17 23:58:57  
Name: startDateTime, dtype: object  
0    2010-06-12 23:57:24  
1    2010-06-12 23:57:24  
2    2010-06-12 23:57:38  
3    2010-06-12 23:57:38  
4    2010-06-12 23:57:40  
5    2010-06-12 23:57:40  
6    2010-06-12 23:57:42  
7    2010-06-12 23:57:42  
8    2010-06-12 23:58:11  
9    2010-06-12 23:58:11  
Name: startDateTime, dtype: datetime64[ns]  
['2010-06-12T23:57:24.000000000' '2010-06-12T23:57:38.000000000'  
'2010-06-12T23:57:40.000000000' ... '2010-06-17T23:58:55.000000000'  
'2010-06-17T23:58:56.000000000' '2010-06-17T23:58:57.000000000']
```

```
stopDateTime:  
count           1416201  
unique          217760  
top      2010-06-13 16:42:25  
freq            9742  
first     2010-06-12 23:59:00  
last      2010-06-17 23:58:59  
Name: stopDateTime, dtype: object  
0    2010-06-13 09:24:52  
1    2010-06-13 09:24:52  
2    2010-06-12 23:59:20  
3    2010-06-12 23:59:20  
4    2010-06-12 23:59:20  
5    2010-06-12 23:59:20  
6    2010-06-12 23:59:22  
7    2010-06-12 23:59:22  
8    2010-06-12 23:59:07  
9    2010-06-12 23:59:07  
Name: stopDateTime, dtype: datetime64[ns]  
['2010-06-13T09:24:52.000000000' '2010-06-12T23:59:20.000000000'
```

```
'2010-06-12T23:59:22.000000000' ... '2010-06-17T23:58:54.000000000'  
'2010-06-17T23:58:57.000000000' '2010-06-17T23:58:55.000000000']
```

```
totalDestinationBytes:  
count      1.416201e+06  
mean      3.489744e+04  
std       7.169916e+05  
min       0.000000e+00  
25%      3.230000e+02  
50%      1.106000e+03  
75%      7.342000e+03  
max      3.955996e+08  
Name: totalDestinationBytes, dtype: float64  
0         0  
1         0  
2        128  
3        128  
4         64  
5         64  
6        128  
7        128  
8         0  
9         0  
Name: totalDestinationBytes, dtype: int64  
[     0    128     64 ... 90547 89145 93232]
```

```
totalDestinationPackets:  
count      1.416201e+06  
mean      3.145587e+01  
std       8.157739e+02  
min       0.000000e+00  
25%      2.000000e+00  
50%      5.000000e+00  
75%      1.100000e+01  
max      5.382280e+05  
Name: totalDestinationPackets, dtype: float64  
0         0  
1         0  
2         2  
3         2  
4         1  
5         1  
6         2  
7         2  
8         0  
9         0
```

APÉNDICE A. MATERIAL ADICIONAL

```
Name: totalDestinationPackets, dtype: int64  
[ 0 2 1 ... 4396 1474 2139]
```

```
totalSourceBytes:  
count    1.416201e+06  
mean     2.822919e+03  
std      9.083839e+05  
min      0.000000e+00  
25%      2.470000e+02  
50%      4.380000e+02  
75%      8.680000e+02  
max      7.632776e+08  
Name: totalSourceBytes, dtype: float64  
0 2633658  
1 2633658  
2 64  
3 64  
4 128  
5 128  
6 128  
7 128  
8 64  
9 64  
Name: totalSourceBytes, dtype: int64  
[2633658 64 128 ... 32844 22041 21958]
```

```
totalSourcePackets:  
count    1.416201e+06  
mean     2.036808e+01  
std      6.724673e+02  
min      0.000000e+00  
25%      3.000000e+00  
50%      6.000000e+00  
75%      1.000000e+01  
max      5.147940e+05  
Name: totalSourcePackets, dtype: float64  
0 28971  
1 28971  
2 1  
3 1  
4 2  
5 2  
6 2  
7 2  
8 1  
9 1
```

```
Name: totalSourcePackets, dtype: int64  
[28971     1     2 ... 3760 3607 915]
```

```
typeAttack:  
count      1416201  
unique       5  
top          N/A  
freq      1349388  
Name: typeAttack, dtype: object  
0      N/A  
1      N/A  
2      N/A  
3      N/A  
4      N/A  
5      N/A  
6      N/A  
7      N/A  
8      N/A  
9      N/A  
Name: typeAttack, dtype: object  
['N/A' 'Infiltration' 'HTTP Dos' 'Distributed DoS' 'Brute Force  
SSH']
```

A.3 Repositorio GIT

<https://git.fic.udc.es/fernando.villar.freire/TFG>

A.4 Selección de atributos

A.4.1 Regresión logística

```
==== Run information ====  
  
Evaluator:    weka.attributeSelection.CfsSubsetEval -P 4 -E 8  
Search:       weka.attributeSelection.GreedyStepwise -R -T  
              -1.7976931348623157E308 -N -1 -num-slots 4  
Relation:     flows-weka.filters.supervised.instance  
              .SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised  
              .attribute.Remove-R3-4,11,16-17,22-weka.filters.unsupervised  
              .attribute.Remove-R19-55-weka.filters.unsupervised  
              .attribute.Remove-R59  
Instances:    133590  
Attributes:   59
```

```
totalDestinationBytes
totalDestinationPackets
sourcePayload
destinationPayload
timeLength
sourceByteRate
destinationByteRate
sourcePacketRate
avgSourcePacketSize
avgDestinationPacketSize
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
protocolName_icmp_ip
protocolName_igmp
protocolName_ip
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_A
sourceTCPFlag_B
sourceTCPFlag_F
sourceTCPFlag_N/A
sourceTCPFlag_P
sourceTCPFlag_R
sourceTCPFlag_S
sourceTCPFlag_U
destinationTCPFlag_A
destinationTCPFlag_B
destinationTCPFlag_F
destinationTCPFlag_N/A
destinationTCPFlag_P
destinationTCPFlag_R
destinationTCPFlag_S
sourcePortResume_wellKnown
sourcePortResume_registered
sourcePortResume_ephemeral
destinationPortResume_wellKnown
destinationPortResume_registered
destinationPortResume_ephemeral
sourceResume_NATServer
sourceResume_external
```

```
sourceResume_internal  
sourceResume_invalid  
sourceResume_mainServer  
sourceResume_private  
sourceResume_secondaryServer  
destinationResume_NATServer  
destinationResume_external  
destinationResume_internal  
destinationResume_invalid  
destinationResume_mainServer  
destinationResume_private  
destinationResume_secondaryServer  
Tag  
Evaluation mode: evaluate on all training data
```

```
==== Attribute Selection on all input data ===
```

Search Method:

Greedy Stepwise (forwards).

Start set: no attributes

Ranking is the order that attributes were added, starting with no attributes. The merit scores in the left column are the goodness of the subset after the adding the corresponding attribute in the right column to the subset.

Attribute Subset Evaluator (supervised, class (nominal): 59 Tag):

CFS Subset Evaluator

Including locally predictive attributes

Ranked attributes:

0.736	53 destinationResume_external
0.634	10 avgDestinationPacketSize
0.603	56 destinationResume_mainServer
0.594	23 protocolName_udp_ip
0.568	6 sourceByteRate
0.544	54 destinationResume_internal
0.528	30 sourceTCPFlag_S
0.517	15 avgSourcePacketSizeDiffMedianScal
0.505	58 destinationResume_secondaryServer
0.494	12 totalDestinationPacketsDiffMedianScal
0.486	52 destinationResume_NATServer
0.477	46 sourceResume_external
0.469	7 destinationByteRate

APÉNDICE A. MATERIAL ADICIONAL

0.463	49 sourceResume_mainServer
0.457	51 sourceResume_secondaryServer
0.451	31 sourceTCPFlag_U
0.445	25 sourceTCPFlag_Blank
0.441	1 totalDestinationBytes
0.436	29 sourceTCPFlag_R
0.431	9 avgSourcePacketSize
0.427	13 timeLengthDiffMedianScal
0.423	22 protocolName_tcp_ip
0.42	14 avgDestinationPacketSizeDiffMedianScal
0.417	43 destinationPortResume_registered
0.413	44 destinationPortResume_ephemeral
0.41	19 protocolName_icmp_ip
0.407	16 destinationByteRateDiffMedianScal
0.404	11 totalDestinationBytesDiffMedianScal
0.401	27 sourceTCPFlag_N/A
0.398	8 sourcePacketRate
0.395	33 destinationTCPFlag_Blank
0.391	2 totalDestinationPackets
0.388	35 destinationTCPFlag_N/A
0.385	18 sourcePacketRateDiffMedianScal
0.383	37 destinationTCPFlag_R
0.38	45 sourceResume_NATServer
0.377	39 sourcePortResume_wellKnown
0.374	17 sourceByteRateDiffMedianScal
0.371	32 destinationTCPFlag_A
0.366	40 sourcePortResume_registered
0.361	42 destinationPortResume_wellKnown
0.356	4 destinationPayload
0.351	5 timeLength
0.345	41 sourcePortResume_ephemeral
0.339	3 sourcePayload
0.333	47 sourceResume_internal
0.326	34 destinationTCPFlag_F
0.317	24 sourceTCPFlag_A
0.309	36 destinationTCPFlag_P
0.301	28 sourceTCPFlag_P
0.293	38 destinationTCPFlag_S
0.261	20 protocolName_igmp
0.237	21 protocolName_ip
0.219	26 sourceTCPFlag_F
0.204	48 sourceResume_invalid
0.191	50 sourceResume_private
0.181	55 destinationResume_invalid
0.171	57 destinationResume_private

A.4. Selección de atributos

Selected attributes: 53,10,56,23,6,54,30,15,58,12,52,46,7,49,51, 31,25,1,29,9,13,22,14,43,44,19,16,11,27,8,33,2,35,18,37,45,39,17, 32,40,42,4,5,41,3,47,34,24,36,28,38,20,21,26,48,50,55,57 : 58

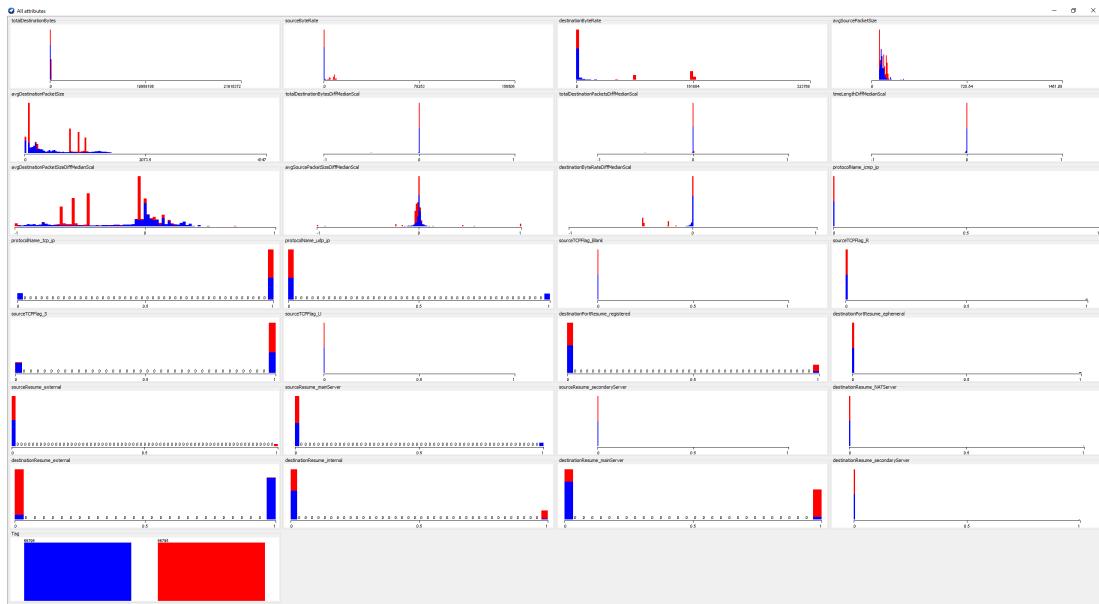


Figura A.1: Resumen visual del dataset resultante para el método de regresión logística

A.4.2 Random forest

```
==== Run information ====
Evaluator:      weka.attributeSelection.InfoGainAttributeEval
Search:        weka.attributeSelection.Ranker -T
              -1.7976931348623157E308 -N -1
Relation:      flows-weka.filters.supervised.instance
              .SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised
              .attribute.Remove-R25-61-weka.filters.unsupervised
              .attribute.Remove-R65
Instances:     133590
Attributes:    65
              totalDestinationBytes
              totalDestinationPackets
              totalSourceBytes
              totalSourcePackets
              sourcePayload
              destinationPayload
              timeLength
              sourceByteRate
```

```
destinationByteRate
sourcePacketRate
destinationPacketRate
avgSourcePacketSize
avgDestinationPacketSize
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
totalSourceBytesDiffMedianScal
totalSourcePacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
destinationPacketRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
protocolName_icmp_ip
protocolName_igmp
protocolName_ip
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_A
sourceTCPFlag_B
sourceTCPFlag_C
sourceTCPFlag_D
sourceTCPFlag_E
sourceTCPFlag_F
sourceTCPFlag_G
sourceTCPFlag_H
sourceTCPFlag_I
sourceTCPFlag_J
sourceTCPFlag_K
sourceTCPFlag_L
sourceTCPFlag_M
sourceTCPFlag_N
sourceTCPFlag_O
sourceTCPFlag_P
sourceTCPFlag_Q
sourceTCPFlag_R
sourceTCPFlag_S
sourceTCPFlag_T
sourceTCPFlag_U
destinationTCPFlag_A
destinationTCPFlag_B
destinationTCPFlag_C
destinationTCPFlag_D
destinationTCPFlag_E
destinationTCPFlag_F
destinationTCPFlag_G
destinationTCPFlag_H
destinationTCPFlag_I
destinationTCPFlag_J
destinationTCPFlag_K
destinationTCPFlag_L
destinationTCPFlag_M
destinationTCPFlag_N
destinationTCPFlag_O
destinationTCPFlag_P
destinationTCPFlag_Q
destinationTCPFlag_R
destinationTCPFlag_S
sourcePortResume_wellKnown
sourcePortResume_registered
sourcePortResume_ephemeral
destinationPortResume_wellKnown
destinationPortResume_registered
destinationPortResume_ephemeral
sourceResume_NATServer
sourceResume_external
sourceResume_internal
sourceResume_invalid
```

```
sourceResume_mainServer
sourceResume_private
sourceResume_secondaryServer
destinationResume_NATServer
destinationResume_external
destinationResume_internal
destinationResume_invalid
destinationResume_mainServer
destinationResume_private
destinationResume_secondaryServer
Tag
Evaluation mode: evaluate on all training data

==== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 65 Tag):
Information Gain Ranking Filter

Ranked attributes:
0.8406333    19 avgDestinationPacketSizeDiffMedianScal
0.8365018    20 avgSourcePacketSizeDiffMedianScal
0.805442     16 totalSourceBytesDiffMedianScal
0.8021542     3 totalSourceBytes
0.7672258    13 avgDestinationPacketSize
0.7630671    15 totalDestinationPacketsDiffMedianScal
0.7552553     1 totalDestinationBytes
0.745353      14 totalDestinationBytesDiffMedianScal
0.7334832    59 destinationResume_external
0.7237885    12 avgSourcePacketSize
0.7204871    17 totalSourcePacketsDiffMedianScal
0.6989378    24 sourcePacketRateDiffMedianScal
0.6970357    23 sourceByteRateDiffMedianScal
0.5773154    21 destinationByteRateDiffMedianScal
0.5668778    22 destinationPacketRateDiffMedianScal
0.539086      8 sourceByteRate
0.5324124      9 destinationByteRate
0.5085535    18 timeLengthDiffMedianScal
0.4926845      2 totalDestinationPackets
0.4752083    11 destinationPacketRate
0.4666739      4 totalSourcePackets
0.4666225    10 sourcePacketRate
```

APÉNDICE A. MATERIAL ADICIONAL

0.3325416	62 destinationResume_mainServer
0.1294923	36 sourceTCPFlag_S
0.1247091	7 timeLength
0.114471	29 protocolName_udp_ip
0.1056558	28 protocolName_tcp_ip
0.1056558	33 sourceTCPFlag_N/A
0.1056558	41 destinationTCPFlag_N/A
0.0931969	60 destinationResume_internal
0.0857159	38 destinationTCPFlag_A
0.0694564	55 sourceResume_mainServer
0.0505303	43 destinationTCPFlag_R
0.049013	49 destinationPortResume_registered
0.0387414	48 destinationPortResume_wellKnown
0.0298672	52 sourceResume_external
0.0146908	6 destinationPayload
0.0133436	47 sourcePortResume_ephemeral
0.0084201	46 sourcePortResume_registered
0.0069968	64 destinationResume_secondaryServer
0.0064677	58 destinationResume_NATServer
0.0056229	53 sourceResume_internal
0.0054301	5 sourcePayload
0.0053789	45 sourcePortResume_wellKnown
0.0046353	30 sourceTCPFlag_A
0.0029679	35 sourceTCPFlag_R
0.0009181	42 destinationTCPFlag_P
0.0007841	50 destinationPortResume_ephemeral
0.000514	34 sourceTCPFlag_P
0.000452	31 sourceTCPFlag_Bank
0.0003474	57 sourceResume_secondaryServer
0.0002977	39 destinationTCPFlag_Bank
0.0001797	37 sourceTCPFlag_U
0.000135	44 destinationTCPFlag_S
0.0000973	51 sourceResume_NATServer
0.0000566	40 destinationTCPFlag_F
0.000052	25 protocolName_icmp_ip
0	54 sourceResume_invalid
0	27 protocolName_ip
0	61 destinationResume_invalid
0	63 destinationResume_private
0	56 sourceResume_private
0	26 protocolName_igmp
0	32 sourceTCPFlag_F

Selected attributes: 19, 20, 16, 3, 13, 15, 1, 14, 59, 12, 17, 24, 23, 21, 22, 8, 9, 18, 2, 11, 4, 10, 62, 36, 7, 29, 28, 33, 41, 60, 38, 55, 43, 49, 48, 52, 6, 47, 46, 64, 58, 53, 5, 45, 30, 35, 42, 50, 34, 31, 57, 39, 37, 44, 51, 40, 25,

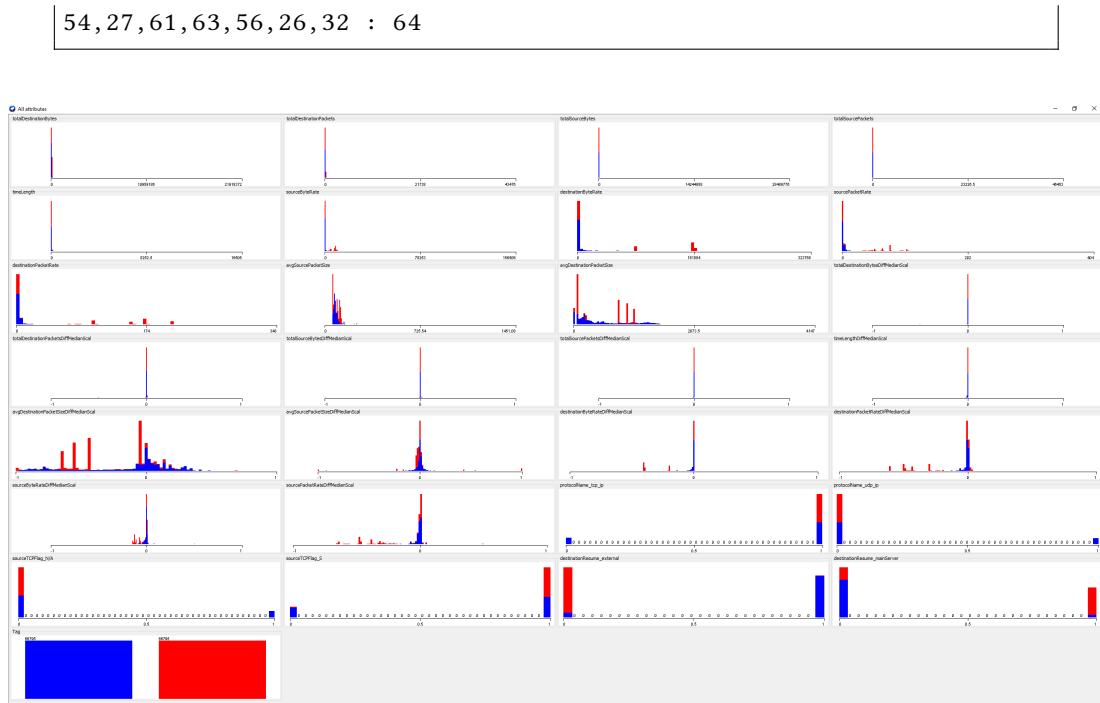


Figura A.2: Resumen visual del dataset resultante para el método de *Random Forest*

A.4.3 SVM

```
== Run information ==

Evaluator: weka.attributeSelection.InfoGainAttributeEval
Search: weka.attributeSelection.Ranker -T
        -1.7976931348623157E308 -N -1
Relation: flows_normalized-weka.filters.supervised.instance
          .SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised
          .attribute.Remove-R26-61-weka.filters.unsupervised
          .attribute.Remove-R66
Instances: 133590
Attributes: 66
           totalDestinationBytes
           totalDestinationPackets
           totalSourceBytes
           totalSourcePackets
           sourcePayload
           destinationPayload
           timeLength
           sourceByteRate
           destinationByteRate
           sourcePacketRate
```

```
destinationPacketRate
avgSourcePacketSize
avgDestinationPacketSize
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
totalSourceBytesDiffMedianScal
totalSourcePacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
destinationPacketRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
appName_BitTorrent
protocolName_icmp_ip
protocolName_igmp
protocolName_ip
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_A
sourceTCPFlag_Blank
sourceTCPFlag_F
sourceTCPFlag_N/A
sourceTCPFlag_P
sourceTCPFlag_R
sourceTCPFlag_S
sourceTCPFlag_U
destinationTCPFlag_A
destinationTCPFlag_Blank
destinationTCPFlag_F
destinationTCPFlag_N/A
destinationTCPFlag_P
destinationTCPFlag_R
destinationTCPFlag_S
sourcePortResume_wellKnown
sourcePortResume_registered
sourcePortResume_ephemeral
destinationPortResume_wellKnown
destinationPortResume_registered
destinationPortResume_ephemeral
sourceResume_NATServer
sourceResume_external
sourceResume_internal
sourceResume_invalid
sourceResume_mainServer
```

```
sourceResume_private
sourceResume_secondaryServer
destinationResume_NATServer
destinationResume_external
destinationResume_internal
destinationResume_invalid
destinationResume_mainServer
destinationResume_private
destinationResume_secondaryServer
Tag
Evaluation mode: evaluate on all training data

==== Attribute Selection on all input data ===

Search Method:
Attribute ranking.

Attribute Evaluator (supervised, Class (nominal): 66 Tag):
Information Gain Ranking Filter

Ranked attributes:
0.8406333    19 avgDestinationPacketSizeDiffMedianScal
0.8365018    20 avgSourcePacketSizeDiffMedianScal
0.8287374    12 avgSourcePacketSize
0.8253313    13 avgDestinationPacketSize
0.805442     16 totalSourceBytesDiffMedianScal
0.7926609    1 totalDestinationBytes
0.7665893    2 totalDestinationPackets
0.7630671    15 totalDestinationPacketsDiffMedianScal
0.7620266    4 totalSourcePackets
0.7580709    3 totalSourceBytes
0.745353     14 totalDestinationBytesDiffMedianScal
0.7334832    60 destinationResume_external
0.7204871    17 totalSourcePacketsDiffMedianScal
0.6989378    24 sourcePacketRateDiffMedianScal
0.6970357    23 sourceByteRateDiffMedianScal
0.5773154    21 destinationByteRateDiffMedianScal
0.5668778    22 destinationPacketRateDiffMedianScal
0.5331584    7 timeLength
0.5085535    18 timeLengthDiffMedianScal
0.4933156    9 destinationByteRate
0.4620452    11 destinationPacketRate
0.4536134    8 sourceByteRate
0.432767     10 sourcePacketRate
```

APÉNDICE A. MATERIAL ADICIONAL

0.3325416	63 destinationResume_mainServer
0.1294923	37 sourceTCPFlag_S
0.114471	30 protocolName_udp_ip
0.1056558	29 protocolName_tcp_ip
0.1056558	42 destinationTCPFlag_N/A
0.1056558	34 sourceTCPFlag_N/A
0.0931969	61 destinationResume_internal
0.0857159	39 destinationTCPFlag_A
0.0694564	56 sourceResume_mainServer
0.0505303	44 destinationTCPFlag_R
0.049013	50 destinationPortResume_registered
0.0387414	49 destinationPortResume_wellKnown
0.0298672	53 sourceResume_external
0.0146908	6 destinationPayload
0.0133436	48 sourcePortResume_ephemeral
0.0084201	47 sourcePortResume_registered
0.0069968	65 destinationResume_secondaryServer
0.0064677	59 destinationResume_NATServer
0.0056229	54 sourceResume_internal
0.0054301	5 sourcePayload
0.0053789	46 sourcePortResume_wellKnown
0.0046353	31 sourceTCPFlag_A
0.0029679	36 sourceTCPFlag_R
0.0023142	25 appName_BitTorrent
0.0009181	43 destinationTCPFlag_P
0.0007841	51 destinationPortResume_ephemeral
0.000514	35 sourceTCPFlag_P
0.000452	32 sourceTCPFlag_Bank
0.0003474	58 sourceResume_secondaryServer
0.0002977	40 destinationTCPFlag_Bank
0.0001797	38 sourceTCPFlag_U
0.000135	45 destinationTCPFlag_S
0.0000973	52 sourceResume_NATServer
0.0000566	41 destinationTCPFlag_F
0.000052	26 protocolName_icmp_ip
0	27 protocolName_igmp
0	64 destinationResume_private
0	62 destinationResume_invalid
0	28 protocolName_ip
0	57 sourceResume_private
0	55 sourceResume_invalid
0	33 sourceTCPFlag_F

Selected attributes: 19, 20, 12, 13, 16, 1, 2, 15, 4, 3, 14, 60, 17, 24, 23, 21, 22, 7, 18, 9, 11, 8, 10, 63, 37, 30, 29, 42, 34, 61, 39, 56, 44, 50, 49, 53, 6, 48, 47, 65, 59, 54, 5, 46, 31, 36, 25, 43, 51, 35, 32, 58, 40, 38, 45, 52, 41, 26, 27, 64, 62, 28,

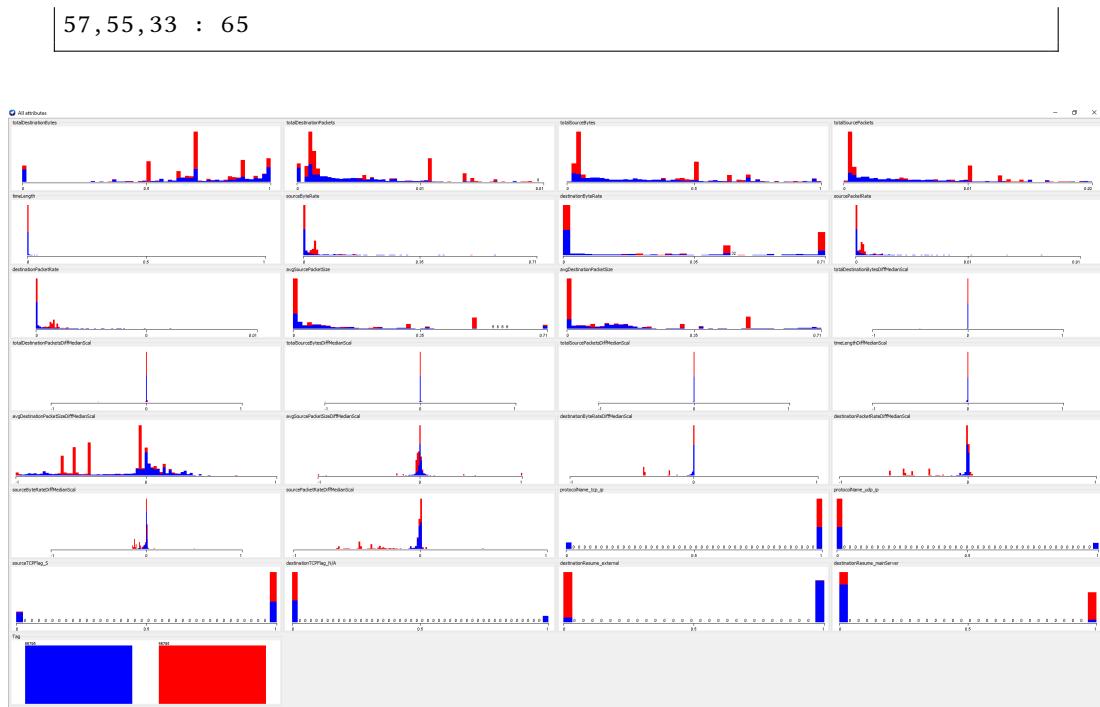


Figura A.3: Resumen visual del dataset resultante para el método de SVM

A.5 Hiperparametrización

A.5.1 Random forest

```
==== Run information ====
Scheme:      weka.classifiers.meta.CVParameterSelection -P "I
              100.0 1300.0 5.0" -P "K 0.0 16.0 5.0" -X 2 -S 1 -W
              weka.classifiers.trees.RandomForest -- -P 100
              -attribute-importance -I 100 -num-slots 8 -K 0 -M 1.0 -V 0.001
              -S 1
Relation:
      flows-weka.filters.supervised.instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervi
Instances:    13359
Attributes:   29
              totalDestinationBytes
              totalDestinationPackets
              totalSourceBytes
              totalSourcePackets
              timeLength
              sourceByteRate
              destinationByteRate
```

```
sourcePacketRate
destinationPacketRate
avgSourcePacketSize
avgDestinationPacketSize
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
totalSourceBytesDiffMedianScal
totalSourcePacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
destinationPacketRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_N/A
sourceTCPFlag_S
destinationResume_external
destinationResume_mainServer
Tag
Test mode: 10-fold cross-validation

==== Classifier model (full training set) ===

Cross-validated Parameter selection.
Classifier: weka.classifiers.trees.RandomForest
Cross-validation Parameter: '-I' ranged from 100.0 to 1300.0 with
    5.0 steps
Cross-validation Parameter: '-K' ranged from 0.0 to 16.0 with 5.0
    steps
Classifier Options: -I 700 -K 12 -P 100 -attribute-importance
    -num-slots 8 -M 1.0 -V 0.001 -S 1

RandomForest

Bagging with 700 iterations and base learner

weka.classifiers.trees.RandomTree -K 12 -M 1.0 -V 0.001 -S 1
    -do-not-check-capabilities

Attribute importance based on average impurity decrease (and number
of nodes using that attribute)

    0.45 ( 1201) destinationResume_external
```

```

0.44 ( 758) destinationResume_mainServer
0.38 ( 6458) totalDestinationBytes
0.38 ( 3137) totalDestinationPackets
0.3 ( 1172) destinationByteRate
0.3 ( 2665) totalSourcePackets
0.3 ( 6797) totalSourceBytes
0.29 ( 2223) timeLength
0.28 ( 2560) sourceByteRate
0.27 ( 4869) avgSourcePacketSize
0.26 ( 581) destinationPacketRate
0.25 ( 348) protocolName_udp_ip
0.24 ( 248) protocolName_tcp_ip
0.23 ( 2564) avgDestinationPacketSize
0.22 ( 185) sourceTCPFlag_N/A
0.22 ( 574) sourceTCPFlag_S
0.22 ( 2872) avgSourcePacketSizeDiffMedianScal
0.21 ( 2178) totalDestinationPacketsDiffMedianScal
0.21 ( 1762) totalSourcePacketsDiffMedianScal
0.21 ( 2744) totalSourceBytesDiffMedianScal
0.2 ( 797) destinationByteRateDiffMedianScal
0.2 ( 2530) totalDestinationBytesDiffMedianScal
0.19 ( 2298) timeLengthDiffMedianScal
0.19 ( 1365) sourcePacketRate
0.19 ( 1886) avgDestinationPacketSizeDiffMedianScal
0.18 ( 519) destinationPacketRateDiffMedianScal
0.16 ( 1184) sourceByteRateDiffMedianScal
0.12 ( 759) sourcePacketRateDiffMedianScal

Time taken to build model: 157.01 seconds

==== Stratified cross-validation ====
==== Summary ====

Correctly Classified Instances      13300          99.5584 %
Incorrectly Classified Instances     59           0.4416 %
Kappa statistic                      0.9912
Mean absolute error                  0.0086
Root mean squared error              0.0634
Relative absolute error              1.7193 %
Root relative squared error         12.6755 %
Total Number of Instances            13359

==== Detailed Accuracy By Class ====

```

APÉNDICE A. MATERIAL ADICIONAL

MCC	ROC Area	TP Rate	FP Rate	Precision		Recall	F-Measure
				PRC Area	Class		
0,991	0,999	0,996	0,005	0,995	0,996	0,996	
				Normal			
		0,995	0,004	0,996	0,995	0,996	
0,991	0,999	0,999		Attack			
Weighted Avg.		0,996	0,004	0,996	0,996	0,996	
		0,991	0,999	0,998			

==== Confusion Matrix ===

```

      a     b    <-- classified as
6655   25 |     a = Normal
      34 6645 |     b = Attack

```

21:07:07: Weka Explorer
21:07:07: (c) 1999-2018 The University of Waikato, Hamilton, New Zealand
21:07:07: web: http://www.cs.waikato.ac.nz/~ml/weka/
21:07:07: Started on sábado, 31 agosto 2019
21:07:21: Base relation is now flows-weka.filters.supervised.
instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised.
attribute.Remove-R25-61-weka.filters.supervised.instance.
StratifiedRemoveFolds-S0-N10-F1-weka.filters.unsupervised.
attribute.Remove-R65-weka.filters.unsupervised.attribute.
Remove-R41,60,38,55,43,49,48,52,6,47,46,64,58,53,5,45,30,35,
42,50,34,31,57,39,37,44,51,40,25,54,27,61,63,56,26,32 (13359
instances)
21:07:48: Started weka.classifiers.meta.CVParameterSelection
21:07:48: Command: weka.classifiers.meta.CVParameterSelection -P "I
100.0 1300.0 5.0" -P "K 0.0 16.0 5.0" -X 2 -S 1 -W
weka.classifiers.trees.RandomForest -- -P 100
-attribute-importance -I 100 -num-slots 8 -K 0 -M 1.0 -V 0.001
-S 1
21:33:23: Finished weka.classifiers.meta.CVParameterSelection

A.5.2 SVM

```

==== Run information ===

Scheme:      weka.classifiers.meta.CVParameterSelection -P "C
              5000.0 10000.0 5.0" -X 2 -S 1 -W weka.classifiers.functions.SMO
              -- -C 10000.0 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K
              "weka.classifiers.functions.supportVector.RBFKernel -C 0 -G
              10.0" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8

```

```
-M -1 -num-decimal-places 4" -do-not-check-capabilities
Relation: flows_normalized-weka.filters.supervised.
instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised.
attribute.Remove-R26-61-weka.filters.unsupervised.attribute
.attribute.Remove-R34,61,39,56,44,50,49,53,6,48,47,65,59,54,5,46,31,36,25,
43,51,35,32,58,40,38,45,52,41,26,27,64,62,28,57,55,
33-weka.filters.supervised.instance.
StratifiedRemoveFolds-S0-N10-F1-weka.filters.
unsupervised.attribute.Remove-R29
Instances: 13359
Attributes: 29
totalDestinationBytes
totalDestinationPackets
totalSourceBytes
totalSourcePackets
timeLength
sourceByteRate
destinationByteRate
sourcePacketRate
destinationPacketRate
avgSourcePacketSize
avgDestinationPacketSize
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
totalSourceBytesDiffMedianScal
totalSourcePacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
destinationPacketRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_S
destinationTCPFlag_N/A
destinationResume_external
destinationResume_mainServer
Tag
Test mode: 10-fold cross-validation

==== Classifier model (full training set) ===

Cross-validated Parameter selection.
Classifier: weka.classifiers.functions.SMO
```

APÉNDICE A. MATERIAL ADICIONAL

```
Cross-validation Parameter: '-C' ranged from 5000.0 to 10000.0 with  
5.0 steps  
Classifier Options: -C 5000 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K  
"weka.classifiers.functions.supportVector.RBFKernel -C 0 -G  
10.0" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8  
-M -1 -num-decimal-places 4" -do-not-check-capabilities  
  
SMO  
  
Kernel used:  
RBF Kernel: K(x,y) = exp(-10.0*(x-y)^2)  
  
Classifier for classes: Normal, Attack  
  
BinarySMO  
  
-----  
  
Number of support vectors: 569  
  
Number of kernel evaluations: 89238120  
  
  
Time taken to build model: 628.2 seconds  
  
==== Stratified cross-validation ====  
==== Summary ====  
  
Correctly Classified Instances      13247          99.1616 %  
Incorrectly Classified Instances     112           0.8384 %  
Kappa statistic                   0.9832  
Mean absolute error                0.0084  
Root mean squared error            0.0916  
Relative absolute error             1.6768 %  
Root relative squared error        18.3127 %  
Total Number of Instances          13359  
  
==== Detailed Accuracy By Class ====  
  
          TP Rate   FP Rate   Precision   Recall    F-Measure  
MCC      ROC Area   PRC Area   Class  
          0,993     0,010     0,990     0,993     0,992  
0,983    0,992     0,987     Normal  
          0,990     0,007     0,993     0,990     0,992  
0,983    0,992     0,988     Attack
```

```

Weighted Avg.      0,992      0,008      0,992      0,992      0,992
          0,983      0,992      0,987

==== Confusion Matrix ====

    a      b  <-- classified as
6634    46 |      a = Normal
    66 6613 |      b = Attack

09:30:11: Weka Explorer
09:30:11: (c) 1999-2018 The University of Waikato, Hamilton, New
Zealand
09:30:11: web: http://www.cs.waikato.ac.nz/~ml/weka/
09:30:11: Started on domingo, 1 septiembre 2019
09:30:29: Base relation is now flows_normalized-weka.filters.
supervised.instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.
unsupervised.attribute.Remove-R26-61-weka.filters.unsupervised.
attribute.Remove-R34,61,39,56,44,50,49,53,6,48,47,65,59,54,5,46,
31,36,25,43,51,35,32,58,40,38,45,52,41,26,27,64,62,28,57,55,33-weka.
filters.supervised.instance.StratifiedRemoveFolds-S0-N10-F1-weka.
filters.unsupervised.attribute.Remove-R29 (13359 instances)
11:28:36: Started weka.classifiers.meta.CVParameterSelection
11:28:36: Command: weka.classifiers.meta.CVParameterSelection -P "C
5000.0 10000.0 5.0" -X 2 -S 1 -W weka.classifiers.functions.SMO
-- -C 10000.0 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K
"weka.classifiers.functions.supportVector.RBFKernel -C 0 -G
10.0" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8
-M -1 -num-decimal-places 4" -do-not-check-capabilities
12:48:36: Finished weka.classifiers.meta.CVParameterSelection

```

A.6 Modelos finales

A.6.1 Regresión logística

```

==== Run information ===

Scheme:      weka.classifiers.functions.SimpleLogistic -I 0 -M 500
           -H 50 -W 0.0
Relation:     flows-weka.filters.supervised.instance.
SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised.
attribute.Remove-R3-4,11,16-17,22-weka.filters.unsupervised.
attribute.Remove-R19-55-weka.filters.unsupervised.attribute.
Remove-R27,8,33,2,35,18,37,45,39,17,32,40,42,4,5,41,3,47,34,
24,36,28,38,20,21,26,48,50,55,57-weka.filters.unsupervised.

```

```
attribute.Remove-R29
Instances:    133590
Attributes:   29
              totalDestinationBytes
              sourceByteRate
              destinationByteRate
              avgSourcePacketSize
              avgDestinationPacketSize
              totalDestinationBytesDiffMedianScal
              totalDestinationPacketsDiffMedianScal
              timeLengthDiffMedianScal
              avgDestinationPacketSizeDiffMedianScal
              avgSourcePacketSizeDiffMedianScal
              destinationByteRateDiffMedianScal
              protocolName_icmp_ip
              protocolName_tcp_ip
              protocolName_udp_ip
              sourceTCPFlag_Blanck
              sourceTCPFlag_R
              sourceTCPFlag_S
              sourceTCPFlag_U
              destinationPortResume_registered
              destinationPortResume_ephemeral
              sourceResume_external
              sourceResume_mainServer
              sourceResume_secondaryServer
              destinationResume_NATServer
              destinationResume_external
              destinationResume_internal
              destinationResume_mainServer
              destinationResume_secondaryServer
              Tag
Test mode:    10-fold cross-validation

==== Classifier model (full training set) ===

SimpleLogistic:

Class Normal :
0.71 +
[destinationByteRate] * -0 +
[protocolName_tcp_ip] * -0.56 +
[protocolName_udp_ip] * 0.93 +
[sourceTCPFlag_S] * -1.11 +
[destinationPortResume_registered] * -0.35 +
[sourceResume_external] * -0.38 +
```

```

[destinationResume_external] * 3.14 +
[destinationResume_internal] * -0.41

Class Attack :
-0.71 +
[destinationByteRate] * 0      +
[protocolName_tcp_ip] * 0.56 +
[protocolName_udp_ip] * -0.93 +
[sourceTCPFlag_S] * 1.11 +
[destinationPortResume_registered] * 0.35 +
[sourceResume_external] * 0.38 +
[destinationResume_external] * -3.14 +
[destinationResume_internal] * 0.41

Time taken to build model: 25.24 seconds

==== Stratified cross-validation ====
==== Summary ====

Correctly Classified Instances      129504          96.9414 %
Incorrectly Classified Instances   4086           3.0586 %
Kappa statistic                   0.9388
Mean absolute error               0.0565
Root mean squared error          0.1598
Relative absolute error          11.3022 %
Root relative squared error     31.9556 %
Total Number of Instances        133590

==== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall    F-Measure
MCC      ROC Area  PRC Area  Class
          0,955    0,017    0,983    0,955    0,969
0,939    0,991    0,990    Normal
          0,983    0,045    0,957    0,983    0,970
0,939    0,991    0,991    Attack
Weighted Avg. 0,969    0,031    0,970    0,969    0,969
          0,939    0,991    0,990

==== Confusion Matrix ===

      a      b  <-- classified as
63815  2980 |      a = Normal
1106  65689 |      b = Attack

```

```
09:10:27: Weka Explorer
09:10:27: (c) 1999-2018 The University of Waikato, Hamilton, New Zealand
09:10:27: web: http://www.cs.waikato.ac.nz/~ml/weka/
09:10:27: Started on domingo, 1 septiembre 2019
09:10:44: Base relation is now flows-weka.filters.supervised.
instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised.
attribute.Remove-R3-4,11,16-17,22-weka.filters.unsupervised.
attribute.Remove-R19-55-weka.filters.unsupervised.attribute.
Remove-R27,8,33,2,35,18,37,45,39,17,32,40,42,4,5,41,3,47,34,
24,36,28,38,20,21,26,48,50,55,57-weka.filters.unsupervised.
attribute.Remove-R29 (133590 instances)
09:11:19: Started weka.classifiers.functions.SimpleLogistic
09:11:19: Command: weka.classifiers.functions.SimpleLogistic -I 0
    -M 500 -H 50 -W 0.0
09:15:07: Finished weka.classifiers.functions.SimpleLogistic
09:27:55: Saved model (09:11:19 - functions.SimpleLogistic) to file
'LR_Final.model'
```

A.6.2 Random forest

```
== Run information ==

Scheme:      weka.classifiers.trees.RandomForest -P 100
             -attribute-importance -I 700 -num-slots 8 -K 12 -M 1.0 -V 0.001
             -S 1
Relation:    flows-weka.filters.supervised.instance.
SpreadSubsample-M1.0-X0.0-S1-weka.filters.unsupervised.
attribute.Remove-R25-61-weka.filters.unsupervised.attribute.
Remove-R65-weka.filters.unsupervised.attribute.Remove-R41,60
,38,55,43,49,48,52,6,47,46,64,58,53,5,45,30,35,42,50,34,31,57
,39,37,44,51,40,25,54,27,61,63,56,26,32
Instances:   133590
Attributes:  29
             totalDestinationBytes
             totalDestinationPackets
             totalSourceBytes
             totalSourcePackets
             timeLength
             sourceByteRate
             destinationByteRate
             sourcePacketRate
             destinationPacketRate
             avgSourcePacketSize
             avgDestinationPacketSize
```

```
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
totalSourceBytesDiffMedianScal
totalSourcePacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
destinationPacketRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_N/A
sourceTCPFlag_S
destinationResume_external
destinationResume_mainServer
Tag
Test mode: 10-fold cross-validation

==== Classifier model (full training set) ====

RandomForest

Bagging with 700 iterations and base learner

weka.classifiers.trees.RandomTree -K 12 -M 1.0 -V 0.001 -S 1
-do-not-check-capabilities

Attribute importance based on average impurity decrease (and number
of nodes using that attribute)

0.37 ( 1638) destinationResume_external
0.33 ( 13212) totalDestinationPackets
0.32 ( 37289) totalDestinationBytes
0.26 ( 12745) destinationByteRate
0.26 ( 9905) totalSourcePackets
0.25 ( 34257) totalSourceBytes
0.24 ( 11356) timeLength
0.23 ( 16787) sourceByteRate
0.22 ( 4871) destinationPacketRate
0.22 ( 235) sourceTCPFlag_N/A
0.21 ( 297) protocolName_tcp_ip
0.21 ( 5949) sourcePacketRate
0.2 ( 18589) avgDestinationPacketSize
0.19 ( 24544) avgSourcePacketSize
```

APÉNDICE A. MATERIAL ADICIONAL

```
0.17 ( 391) protocolName_udp_ip
0.16 ( 1601) sourceTCPFlag_S
0.16 ( 13405) avgDestinationPacketSizeDiffMedianScal
0.16 ( 1688) destinationResume_mainServer
0.15 ( 11672) totalDestinationBytesDiffMedianScal
0.15 ( 7164) destinationByteRateDiffMedianScal
0.15 ( 8101) totalDestinationPacketsDiffMedianScal
0.15 ( 16418) avgSourcePacketSizeDiffMedianScal
0.14 ( 15181) totalSourceBytesDiffMedianScal
0.14 ( 7140) timeLengthDiffMedianScal
0.14 ( 5822) totalSourcePacketsDiffMedianScal
0.13 ( 3846) destinationPacketRateDiffMedianScal
0.12 ( 7684) sourceByteRateDiffMedianScal
0.1 ( 3677) sourcePacketRateDiffMedianScal
```

Time taken to build model: 287.52 seconds

```
==== Stratified cross-validation ====
==== Summary ====


```

Correctly Classified Instances	133197	99.7058 %
Incorrectly Classified Instances	393	0.2942 %
Kappa statistic	0.9941	
Mean absolute error	0.0052	
Root mean squared error	0.0498	
Relative absolute error	1.0358 %	
Root relative squared error	9.9612 %	
Total Number of Instances	133590	

```
==== Detailed Accuracy By Class ===


```

		TP Rate	FP Rate	Precision	Recall	F-Measure
MCC	ROC Area	PRC Area	Class			
		0,997	0,003	0,997	0,997	0,997
0,994	1,000	1,000	Normal			
		0,997	0,003	0,997	0,997	0,997
0,994	1,000	1,000	Attack			
Weighted Avg.	0,997	0,003	0,997	0,997	0,997	0,997
	0,994	1,000	1,000			

```
==== Confusion Matrix ===


```

a	b	<- classified as
66626	169	a = Normal
224	66571	b = Attack

```

03:50:19: Weka Explorer
03:50:19: (c) 1999-2018 The University of Waikato, Hamilton, New
Zealand
03:50:19: web: http://www.cs.waikato.ac.nz/~ml/weka/
03:50:19: Started on domingo, 1 septiembre 2019
03:50:35: Base relation is now
    flows-weka.filters.supervised.instance.
SpreadSubsample-M1.0-X0.0-S1-weka.filters.
unsupervised.attribute.Remove-R25-61-weka.
filters.unsupervised.attribute.Remove-R65-weka.
filters.unsupervised.attribute.Remove-R41,60,38,
55,43,49,48,52,6,47,46,64,58,53,5,45,30,35,42,50,
34,31,57,39,37,44,51,40,25,54,27,61,63,56,26,32 (133590 instances)
03:51:09: Started weka.classifiers.trees.RandomForest
03:51:09: Command: weka.classifiers.trees.RandomForest -P 100
    -attribute-importance -I 700 -num-slots 8 -K 12 -M 1.0 -V 0.001
    -S 1
04:37:47: Finished weka.classifiers.trees.RandomForest
09:08:31: Saved model (03:51:09 - trees.RandomForest) to file
    'RF_Final.model'

```

A.6.3 SVM

```

==== Run information ===

Scheme:      weka.classifiers.meta.CVParameterSelection -P "C
              5000.0 10000.0 5.0" -X 2 -S 1 -W weka.classifiers.functions.SMO
              -- -C 10000.0 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K
              "weka.classifiers.functions.supportVector.RBFKernel -C 0 -G
              10.0" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8
              -M -1 -num-decimal-places 4" -do-not-check-capabilities
Relation:     flows_normalized-weka.filters.supervised.
instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.
unsupervised.attribute.Remove-R26-61-weka.filters.
unsupervised.attribute.Remove-R34,61,39,56,44,50,49
              ,53,6,48,47,65,59,54,5,46,31,36,25,43,51,35,32,58,40
              ,38,45,52,41,26,27,64,62,28,57,55,33-weka.filters
              .supervised.instance.StratifiedRemoveFolds-S0-N10-F1-weka
              .filters.unsupervised.attribute.Remove-R29
Instances:    13359
Attributes:   29
              totalDestinationBytes
              totalDestinationPackets
              totalSourceBytes

```

```
totalSourcePackets
timeLength
sourceByteRate
destinationByteRate
sourcePacketRate
destinationPacketRate
avgSourcePacketSize
avgDestinationPacketSize
totalDestinationBytesDiffMedianScal
totalDestinationPacketsDiffMedianScal
totalSourceBytesDiffMedianScal
totalSourcePacketsDiffMedianScal
timeLengthDiffMedianScal
avgDestinationPacketSizeDiffMedianScal
avgSourcePacketSizeDiffMedianScal
destinationByteRateDiffMedianScal
destinationPacketRateDiffMedianScal
sourceByteRateDiffMedianScal
sourcePacketRateDiffMedianScal
protocolName_tcp_ip
protocolName_udp_ip
sourceTCPFlag_S
destinationTCPFlag_N/A
destinationResume_external
destinationResume_mainServer
Tag
Test mode: 10-fold cross-validation

==== Classifier model (full training set) ====

Cross-validated Parameter selection.
Classifier: weka.classifiers.functions.SMO
Cross-validation Parameter: '-C' ranged from 5000.0 to 10000.0 with
5.0 steps
Classifier Options: -C 5000 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K
"weka.classifiers.functions.supportVector.RBFKernel -C 0 -G
10.0" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8
-M -1 -num-decimal-places 4" -do-not-check-capabilities

SMO

Kernel used:
RBF Kernel: K(x,y) = exp(-10.0*(x-y)^2)

Classifier for classes: Normal, Attack
```

```
BinarySMO
-----
Number of support vectors: 569
Number of kernel evaluations: 89238120
Time taken to build model: 628.2 seconds
==== Stratified cross-validation ====
==== Summary ===

Correctly Classified Instances      13247      99.1616 %
Incorrectly Classified Instances    112       0.8384 %
Kappa statistic                   0.9832
Mean absolute error               0.0084
Root mean squared error          0.0916
Relative absolute error           1.6768 %
Root relative squared error     18.3127 %
Total Number of Instances        13359

==== Detailed Accuracy By Class ===

      TP Rate   FP Rate   Precision   Recall   F-Measure
MCC      ROC Area   PRC Area   Class
          0,993     0,010     0,990     0,993     0,992
  0,983     0,992     0,987     Normal
          0,990     0,007     0,993     0,990     0,992
  0,983     0,992     0,988     Attack
Weighted Avg.    0,992     0,008     0,992     0,992     0,992
  0,983     0,992     0,987

==== Confusion Matrix ====

    a     b  <-- classified as
6634   46 |     a = Normal
    66 6613 |     b = Attack

09:30:11: Weka Explorer
09:30:11: (c) 1999-2018 The University of Waikato, Hamilton, New Zealand
09:30:11: web: http://www.cs.waikato.ac.nz/~ml/weka/
09:30:11: Started on domingo, 1 septiembre 2019
```

APÉNDICE A. MATERIAL ADICIONAL

```
09:30:29: Base relation is now flows_normalized-weka.filters.  
supervised.instance.SpreadSubsample-M1.0-X0.0-S1-weka.filters.  
unsupervised.attribute.Remove-R26-61-weka.filters.unsupervised.  
attribute.Remove-R34,61,39,56,44,50,49,53,6,48,47,65,59,54,5,46,  
31,36,25,43,51,35,32,58,40,38,45,52,41,26,27,64,62,28,57,55,33-weka.  
filters.supervised.instance.StratifiedRemoveFolds-S0-N10-F1-weka.  
filters.unsupervised.attribute.Remove-R29 (13359 instances)  
11:28:36: Started weka.classifiers.meta.CVParameterSelection  
11:28:36: Command: weka.classifiers.meta.CVParameterSelection -P "C  
5000.0 10000.0 5.0" -X 2 -S 1 -W weka.classifiers.functions.SMO  
-- -C 10000.0 -L 0.001 -P 1.0E-12 -N 2 -V -1 -W 1 -K  
"weka.classifiers.functions.supportVector.RBFKernel -C 0 -G  
10.0" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8  
-M -1 -num-decimal-places 4" -do-not-check-capabilities  
12:48:36: Finished weka.classifiers.meta.CVParameterSelection
```


Lista de acrónimos

IDS *Intrusion detection system.*

ML *Machine learning.*

NTA *Network traffic analysis.*

SVM *Support vector machine.*

CRISP-DM *Cross-industry standard process for data mining.*

TI *Tecnologías de la información.*

IA *Inteligencia artificial.*

NAT *Network address translation.*

KDD *Knowledge discovery in database.*

TCP *Transmission control protocol.*

SNMP *Simple network management protocol.*

UDP *User Datagram Protocol.*

SCTP *Stream control transmission protocol.*

IETF *Internet engineering task force.*

SQL *Structured query language.*

GUI *Graphical user interface.*

CLI *Command-line interface.*

AWS *Amazon web services.*

HTTP *Hypertext transfer protocol.*

HTTPS *Hypertext transfer protocol secure.*

SSH *Secure shell.*

ICMP *Internet control message protocol.*

DPI *Deep packet inspection.*

IQR *Interquartile range.*

DBFS *Databricks file system.*

RDD *Resilient data distributed.*

Bibliografía

- [1] C. S. Inc., “Cisco visual networking index: Forecast and trends, 2017–2022 white paper,” 2019. [Online]. Available: <https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html>
- [2] L. Columbus, “Roundup of cloud computing forecasts and market estimates, 2018,” 2019. [En línea]. Disponible en: <https://www.forbes.com/sites/louiscolumbus/2018/09/23/roundup-of-cloud-computing-forecasts-and-market-estimates-2018/#7e6561c507b0>
- [3] G. Golomb, “Is nta just another kind of ids?” 2019. [En línea]. Disponible en: <https://securityboulevard.com/2019/01/is-nta-just-another-kind-of-ids/>
- [4] K. Panetta, “The gartner it security approach for the digital age,” 2017. [En línea]. Disponible en: <https://www.gartner.com/smarterwithgartner/the-gartner-it-security-approach-for-the-digital-age/>
- [5] J. Schreiber, “Open source ids tools: Comparing suricata, snort, bro (zeek), linux,” 2018. [En línea]. Disponible en: <https://www.alienvault.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview>
- [6] Gartner, “Market guide for network traffic analysis,” 2019. [En línea]. Disponible en: <https://www.gartner.com/en/documents/3902353/market-guide-for-network-traffic-analysis>
- [7] T. Dunning and E. Friedman, *Practical Machine Learning: A New Look at Anomaly Detection*, 1st ed. O'Reilly, 2014.
- [8] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, Apr 2014.

- [9] B. Claise, B. Trammell, and P. Aitken, “Specification of the ip flow information export (ipfix) protocol for the exchange of flow information (rfc7011),” 2013. [En línea]. Disponible en: <https://tools.ietf.org/html/rfc7011>
- [10] J. Steinberger, L. Schehlmann, S. Abt, and H. Baier, “Anomaly detection and mitigation at internet scale: A survey,” in *Emerging Management Mechanisms for the Future Internet*, G. Doyen, M. Waldburger, P. Čeleda, A. Sperotto, and B. Stiller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 49–60.
- [11] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, “Flow monitoring explained: From packet capture to data analysis with netflow and ipfix,” *IEEE COMMUNICATION SURVEYS TUTORIALS*, vol. 16, no. 4, 2014.
- [12] G. Roscigno, “The role of distributed computing in big data science: Case studies in forensics and bioinformatics,” 2015.
- [13] P. Haag, “nfdump,” 2014. [En línea]. Disponible en: <https://github.com/phaag/nfdump>
- [14] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, *From Data Mining to Knowledge Discovery in Databases*, 1st ed. AIMag, vol. 17, no. 3, p. 37, 1996.
- [15] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth, *CRISP-DM 1.0 Step-by-step data mining guides*. 1, 2000.
- [16] C.-D. consortium, “About crisp-dm,” 2000. [En línea]. Disponible en: <http://crisp-dm.eu/home/about-crisp-dm/>
- [17] J. Hurwitz and D. Kirsch, *Machine Learning For Dummies, IBM Limited Edition*, 1st ed. John Wiley Sons, Inc., 2018.
- [18] A. Kumar, *Learning Predictive Analytics with Python*, 1st ed. Packt, 2016.
- [19] J. Brownlee, “Bagging and random forest ensemble algorithms for machine learning,” 2016. [En línea]. Disponible en: <https://machinelearningmastery.com/bagging-and-random-forest-ensemble-algorithms-for-machine-learning/>
- [20] --, “Support vector machines for machine learning,” 2016. [En línea]. Disponible en: <https://machinelearningmastery.com/support-vector-machines-for-machine-learning/>
- [21] ElevenPaths, “Netflow, machine learning y la detección de anomalías en red: una aproximación académica (parte i),” 2017. [En línea]. Disponible en: <https://empresas.blogthinkbig.com/netflow-machine-learning-y-la-deteccion-2/>

BIBLIOGRAFÍA

- [22] C. Systems, “Cisco systems netflow services export version 9,” 2004. [En línea]. Disponible en: <https://tools.ietf.org/html/rfc3954>
- [23] J. Quittek, T. Zseby, B. Claise, and S. Zander, “Requirements for ip flow information export (ipfix),” 2004. [En línea]. Disponible en: <https://tools.ietf.org/html/rfc3917>
- [24] Suricata, “Suricata.” [En línea]. Disponible en: <https://suricata-ids.org/>
- [25] Snort, “Snort.” [En línea]. Disponible en: <https://www.snort.org/>
- [26] G. Golomb, “Is nta just another kind of ids?” 2019. [En línea]. Disponible en: <https://securityboulevard.com/2019/01/is-nta-just-another-kind-of-ids/>
- [27] T. Liu, “Network traffic analysis (nta),” 2019. [En línea]. Disponible en: <https://www.cyberdefensemagazine.com/network-traffic-analysis-nta/>
- [28] J. Laskowski, “The internals of apache spark 2.4.2,” 2017. [En línea]. Disponible en: <https://jaceklaskowski.gitbooks.io/mastering-apache-spark/>
- [29] A. Luashchuk, “Why i think python is perfect for machine learning and artificial intelligence,” 2019. [En línea]. Disponible en: <https://towardsdatascience.com/8-reasons-why-python-is-good-for-artificial-intelligence-and-machine-learning-4a23f6bed2e6>
- [30] A. Menzinsky, G. López, and J. Palacio, *Guía de Scrum Manager: Manual para consulta y formación de Scrum Manager*, 1st ed. Iubaris Info 4 Media SL, 2016.
- [31] C. I. for Cybersecurity, “Intrusion detection evaluation dataset (iscxids2012),” 2012. [En línea]. Disponible en: <https://www.unb.ca/cic/datasets/ids.html>
- [32] 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, Issue 3, p. 357-374, 2012.
- [33] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, *Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization*. 4th International Conference on Information Systems Security and Privacy (ICISSP)), pages 108-116, 2018.
- [34] O. Niculaescu, “Netflow data generation with nfdump and softflowd,” 2015. [En línea]. Disponible en: <https://elf11.github.io/2015/09/10/NetFlows-data-generation.html>
- [35] I. S. I. U. of Southern California, “Transmission control protocol darpa internet program protocol specification,” 1981. [En línea]. Disponible en: <https://tools.ietf.org/html/rfc793>

- [36] S. BLOG, “Top 20 and 200 most scanned ports in the cybersecurity industry,” 2007. [En línea]. Disponible en: <https://securitytrails.com/blog/top-scanned-ports>
- [37] W. Badr, “5 ways to detect outliers/anomalies that every data scientist should know (python code),” 2005. [En línea]. Disponible en: <https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623>
- [38] E. Lewinson, “Outlier detection with isolation forest,” 2005. [En línea]. Disponible en: <https://towardsdatascience.com/outlier-detection-with-isolation-forest-3d190448d45e>
- [39] J. H. Orallo, M. J. R. Quintana, and C. F. Ramírez, *Introducción a la minería de datos*, 1st ed. Pearson, 2004.
- [40] scikit learn.org, “Compare the effect of different scalers on data with outliers,” 2007-2019. [En línea]. Disponible en: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-glr-auto-examples-preprocessing-plot-all-scaling-py
- [41] I. E. T. Force, “Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry,” 2011. [En línea]. Disponible en: <https://tools.ietf.org/html/rfc6335>
- [42] T. Afonja, “Accuracy paradox,” 2017. [En línea]. Disponible en: <https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b>
- [43] J. Brownlee, “8 tactics to combat imbalanced classes in your machine learning dataset,” 2015. [En línea]. Disponible en: <https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset>
- [44] E. Frank, “Oversampling and undersampling,” 2019. [En línea]. Disponible en: <https://waikato.github.io/weka-blog/posts/2019-01-30-sampling/>
- [45] I. H. Witten, F. Eibe, and M. A. Hall, *Data mining - Practical Machine Learning Tools and Techniques*, 3rd ed. Elsevier, 2011.
- [46] S. T., “Entropy: How decision trees make decisions,” 2011. [En línea]. Disponible en: <https://towardsdatascience.com/entropy-how-decision-trees-make-decisions-2946b9c18c8>
- [47] A. C. Müller and S. Guido, *Introduction to Machine Learning with Python - A Guide for Data Scientists*, 1st ed. O'Reilly, 2016.

BIBLIOGRAFÍA

- [48] P. R. de los Santos, “Machine learning a tu alcance: La matriz de confusión,” 2018. [En línea]. Disponible en: <https://empresas.blogthinkbig.com/ml-a-tu-alcance-matriz-confusion/>
- [49] F. Weka, “Logistic vs simple logistic,” 2014. [En línea]. Disponible en: <https://weka.8497.n7.nabble.com/Logistic-VS-Simple-Logistic-td31410.html>
- [50] C. Lee, “Feature importance measures for tree models — part i,” 2017. [En línea]. Disponible en: <https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3>
- [51] Databricks, “Databricks file system,” 2019. [En línea]. Disponible en: <https://docs.databricks.com/user-guide/databricks-file-system.html>
- [52] D. Weka, “Spark sql, dataframes and datasets guide,” 2019. [En línea]. Disponible en: <https://spark.apache.org/docs/latest/sql-programming-guide.html>
- [53] M. Cermak, D. Tovarňák, M. Lastovicka, and P. Celeda, “A performance benchmark for netflow data analysis on distributed stream processing systems,” 04 2016, pp. 919–924.
- [54] B. Chambers and M. Armbrust, “Taking apache spark’s structured streaming to production,” 2017. [En línea]. Disponible en: <https://databricks.com/blog/2017/05/18/taking-apache-sparks-structured-structured-streaming-to-production.html>
- [55] Apache, “Structured streaming programming guide,” 2017. [En línea]. Disponible en: <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>
- [56] A. Spark, “Tuning spark.” [En línea]. Disponible en: <https://spark.apache.org/docs/latest/tuning.html#memory-management-overview>

