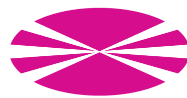


Facultade de Informática
UNIVERSIDADE DA CORUÑA

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

Uso de algoritmos de aprendizaje máquina para la clasificación de tráfico de red

Estudiante: Christian Manuel Varela Álvarez
Dirección: Diego Fernández Iglesias
Dirección: Francisco Javier Nóvoa Manuel

A Coruña, setembro de 2020.

A mi padre, mi ángel de la guarda

Agradecimientos

Quiero agradecer a mi familia el apoyo que me han ofrecido y el esfuerzo que han hecho para que yo haya podido estudiar esta carrera y así tener una profesión con la que disfruto.

También agradecimientos a Cristal, por estar hasta en los días más oscuros apoyándome y animándome.

A mis amigos y amigas, por estar siempre ahí.

A todos, gracias de todo corazón.

Resumen

La cuarentena ocasionada por la pandemia mundial a causa del virus COVID-19 ha mostrado el camino que está tomando la sociedad en la que vivimos, una sociedad donde la filosofía del siempre conectado toma más fuerza que nunca. Muestra de esto ha sido el crecimiento del teletrabajo, provocando el aumento del tráfico de red a unas velocidades vertiginosas, y con ello el número de ataques y de nuevas amenazas en el mundo cibernético.

Ante este escenario surge la necesidad de mejorar y actualizar los planes de defensa. Para poder analizar la gran cantidad de tráfico de red que se genera, aparece la estrategia de agregación de flujos, permitiendo agrupar el tráfico en una serie de paquetes que comparten unos valores concretos y así poder reducir la cantidad de datos a analizar mientras se conserva toda la información necesaria para dicha tarea.

Aún así, esta agregación no consigue reducir lo suficiente las cantidades de datos, por lo que es aquí donde entra en juego el *Big Data*, que con el uso de robustas herramientas de *Machine Learning* junto con los sistemas distribuidos, permiten acometer la tarea de clasificación de tráfico de red de forma sencilla, eficiente y escalable.

Este es el tema que aborda este proyecto, donde aplicamos minería de datos sobre un conjunto de flujos de red para que mediante la selección de tres algoritmos de clasificación de aprendizaje máquina poder crear tres modelos que son capaces de predecir si un flujo es tráfico normal o de ataque. Para esto, seguimos las fases marcadas por la metodología CRISP-DM.

Finalmente, cada uno de estos modelos los desplegamos de forma distribuida para poder ver la importancia que tienen los sistemas distribuidos para el análisis en tiempo real del tráfico de una red.

Abstract

The quarantine caused by the global pandemic due to the COVID-19 virus has shown the path that the society we live in is taking, a society where the philosophy of the always connected is taking on more strength than ever. An example of this has been the growth of teleworking, causing network traffic to increase at dizzying speeds, and with it the number of attacks and new threats in the cyber world.

In view of this scenario, the need to improve and update defence plans arises. In order to analyse the large amount of network traffic generated, a strategy of flow aggregation appears, allowing traffic to be grouped into a series of packets that share specific values and thus reducing the amount of data to be analysed while conserving all the information necessary for this task.

Even so, this aggregation does not manage to reduce the amounts of data sufficiently, so this is where Big Data comes into play. With the use of robust Machine Learning tools together with distributed systems, this allows the task of classifying network traffic to be undertaken in a simple, efficient and scalable manner.

This is the subject that we address per project, where we apply data mining on a set of network flows so that by selecting three machine learning classification algorithms we can create three models that are able to predict whether a flow is normal or attack traffic. For this, we follow the phases marked by the CRISP-DM methodology.

Finally, each one of these models is deployed in a distributed way in order to see the importance of distributed systems for the real-time analysis of network traffic.

Palabras clave:

- Machine Learning
- Big Data
- Flujo
- Anomalía
- Sistema distribuido
- Clasificación
- Cluster
- Regresión Logística
- Random Forest
- Naive Bayes

Keywords:

- Machine Learning
- Big Data
- Flow
- Anomaly
- Distributed system
- Classification
- Cluster
- Logistic Regression
- Random Forest
- Naive Bayes

Índice general

1 Introducción **1**

1.1 Algoritmos de aprendizaje máquina y ciberseguridad 1

1.2 Tráfico normal y anómalo 2

1.3 El *Big Data* y cómo influye en el tráfico de red 2

1.3.1 Computación distribuida 3

1.4 Objetivos 5

1.5 Repositorio Git 5

2 Metodología **7**

2.1 CRISP-DM 7

3 Comprensión del negocio **11**

3.1 *Machine Learning* 11

3.1.1 El objetivo principal 12

3.1.2 Tipo de aprendizaje 12

3.2 Algoritmos 13

3.2.1 Regresión Logística 14

3.2.2 *Random Forest* 15

3.2.3 *Naive Bayes* 17

3.3 Python y PySpark 18

3.4 Scikit-Learn 18

3.5 Apache Spark y Databricks 19

3.6 Apache Kafka 21

3.7 Planificación 23

4 Comprensión de los datos **29**

4.1 El *dataset* 29

4.2 Importación de los datos 30

4.3 Exploración 30

5 Preparación de los datos **35**

5.1 Limpieza de los datos 35

5.2 Transformación 38

5.2.1 Creación de características 38

5.2.2 Escalado de los datos 40

5.3 Selección de las características 41

5.3.1 Regresión Logística 43

5.3.2 *Random Forest* 45

5.3.3 *Naive Bayes* 45

6 Modelado **49**

6.1 Validación 49

6.1.1 Métricas de rendimiento 49

6.1.2 Validación cruzada 51

6.1.3 Resultado de las métricas 52

6.2 Hiperparametrización 54

6.2.1 Regresión Logística 55

6.2.2 *Random Forest* 55

6.2.3 *Naive Bayes* 56

7 Evaluación **57**

8 Despliegue **59**

8.1 Pruebas de rendimiento 61

9 Conclusiones y líneas futuras **69**

Lista de acrónimos **71**

Bibliografía **73**

Índice de figuras

1.1 Diagrama de un sistema distribuido y un sistema en paralelo. 4
1.2 Breve diagrama del funcionamiento de *MapReduce*. 4

2.1 Los cuatro niveles de abstracción de CRISP-DM. 8
2.2 Las seis fases iterativas de la metodología CRISP-DM. 8

3.1 Algoritmos implementados en MLlib [1]. 13
3.2 Plantilla de Scikit-Learn para la elección de los algoritmos. 14
3.3 Fórmula de la función sigmoidea y su gráfica. 15
3.4 Ejemplo de Árbol de Decisión [2]. 16
3.5 Fórmula del teorema de Bayes [3]. 17
3.6 Arquitectura de Apache Spark [4]. 19
3.7 Instalación de Apache Spark en modo *Standalone* [5]. 20
3.8 Instalación de Apache Spark con gestor de recursos [5]. 21
3.9 Pantalla de configuración de un *cluster* en Databricks. 22
3.10 Diagrama del funcionamiento de Kafka. 23
3.11 Estructura de las iniciativas en la gestión ágil de proyectos. 26

4.1 Diagrama del escenario del *dataset* UNB ISCX IDS 2012. 30
4.2 Información de las características iniciales del *dataset*. 31
4.3 Cantidad de flujos normales y de ataques. 32
4.4 Información la característica *sourceTCPFlagsDescription*. 33

5.1 Ejemplo de diagrama de cajas. 35
5.2 Diferencia entre identificar una muestra normal (izquierda) y una anómala (derecha). 36
5.3 Matriz de correlaciones con algunas de las principales característica. 42

6.1 Matriz de confusión. 50

6.2 Validación cruzada [6]. 52
6.3 Matriz de Confusión del modelo de Regresión Logística. 53
6.4 Matriz de Confusión del modelo de *Random Forest*. 53
6.5 Matriz de Confusión del modelo de *Naive Bayes*. 54

8.1 Métricas de rendimiento para *Naive Bayes* distribuido con dos trabajadores. . . 62
8.2 Métricas de rendimiento para *Naive Bayes* distribuido con cuatro trabajadores. 63
8.3 Métricas de rendimiento para *Naive Bayes* distribuido con ocho trabajadores. . 64
8.4 Métricas de rendimiento del *cluster* para *Random Forest* distribuido con dos trabajadores. 65
8.5 Métricas de rendimiento del *cluster* para *Random Forest* distribuido con cuatro trabajadores. 65
8.6 Métricas de rendimiento del *cluster* para *Random Forest* distribuido con ocho trabajadores. 66
8.7 Distribución de la carga de trabajo del *cluster* para *Random Forest* distribuido con dos nodos trabajadores. 66
8.8 Distribución de la carga de trabajo del *cluster* para *Random Forest* distribuido con cuatro nodos trabajadores. 66
8.9 Distribución de la carga de trabajo del *cluster* para *Random Forest* distribuido con ocho nodos trabajadores. 67

Índice de tablas

3.1 Planificación del proyecto. 27

5.1 Resumen de las características seleccionadas para Regresión Logística y su importancia. 43
5.2 Resumen de las características seleccionadas para *Random Forest* y su importancia. 46
5.3 Resumen de las características seleccionadas para Gaussian Naive Bayes y su importancia. 48
5.4 Resumen de las características seleccionadas para Bernoulli Naive Bayes y su importancia. 48

6.1 Resultados de los tres modelos 52
6.2 Resultados de los tres modelos hiperparametrizados 55

7.1 Resultados de los tres modelos tempranos 58

8.1 Comparación de resultados entre Scikit-Learn y Spark. 60
8.2 Métricas de Regresión Logística. 62
8.3 Métricas de *Random Forest*. 62
8.4 Métricas de *Naive Bayes* (Bernoulli). 63

Capítulo 1
Introducción

Nuestra sociedad está viviendo desde hace años el comienzo de una nueva era, llamada la ”Era Digital”, en la que todo gira en torno a las nuevas tecnologías y a Internet, y en la que hoy más que nunca estamos todos conectados con todos, lo cual está cambiando nuestras costumbres y nuestras vidas. Muestra de esto son los términos Internet, nube o *cloud*, Inteligencia Artificial, *Big Data*, ciberseguridad, etc., términos que se han convertido en nuestro día a día y en los titulares de los medios de comunicación.

Esta nueva era hace que el tráfico de red lleve desde unos años creciendo a una altísima velocidad, lo que se traduce en grandes cantidades de datos a analizar a la hora de mantener una red, y es en este sentido por donde se enfoca este trabajo, donde a partir de una gran cantidad de datos veremos qué técnicas se pueden utilizar para facilitar el análisis de red y poder realizar tareas de clasificación.

1.1 Algoritmos de aprendizaje máquina y ciberseguridad

En los últimos años el área de la ciberseguridad ha experimentado un enorme crecimiento, tanto en el número de ataques como en la variedad de las amenazas, y debido al mundo hiperconectado en el que vivimos la ciberseguridad tiene un papel cada vez más protagonista.

De entre las tareas a realizar ante un ciberataque, prevención y detección son las dos en las que más fácil se puede ver el uso de *Machine Learning* (ML), donde como ejemplo se pueden aplicar técnicas de clasificación para identificar los tipos de tráfico de red.

Hay dos métodos muy usados para identificar el tráfico de una red: Sistemas de Detección de Intrusos (IDS) y Sistemas de Prevención de Instrusos (IPS). En cuanto a los IDS se pueden clasificar en dos categorías: basados en heurísticas y basados en reglas. Los primeros utilizan estructuras de *Machine Learning* de alto nivel para verificar el comportamiento del tráfico y cuando haya una anomalía en el comportamiento habitual poder detectarlo. Los segundos funcionan con unas reglas basadas en definiciones de vulnerabilidades conocidas que

se consideran como ataques.

1.2 Tráfico normal y anómalo

La definición de tráfico de red normal puede ser tan vaga y simple como aquel tráfico que tiene unas características que son las esperadas. Es por esto por lo que una anomalía la podríamos definir como un comportamiento inesperado de acuerdo al funcionamiento normal de la red.

Dentro de las anomalías podemos diferenciar las que se producen por algún fallo o comportamiento extraño de algún componente *software* o *hardware*, de las anomalías que se producen por algún tipo de ataque.

Lo que caracteriza a una anomalía es que previamente no sabemos cómo es, solo lo sabemos llegado el punto de compararla con lo que sabemos que es normal.

Este es uno de los aspectos en los que se guía el trabajo, donde gracias al uso de los algoritmos de *Machine Learning* se pueden crear patrones y generalizar el comportamiento normal de la red, para posteriormente usar esos patrones en la clasificación del tráfico de red detectando como anómalo lo que no se ciña a lo normal.

1.3 El *Big Data* y cómo influye en el tráfico de red

En esta era en la que todos estamos conectados con todos y con todo, la cantidad de datos que se mueven por la red cada vez es más grande, rozando cifras tan altas como los 4.416.720 GB que de media se movieron en Estados Unidos cada sesenta segundos en 2019, o las 694.444 horas de vídeo que se reproducen en Netflix al mismo tiempo [7].

Cifras vertiginosas que siguen creciendo y que nos sirven como introducción a la definición de *Big Data*: conjuntos de datos o combinaciones de conjuntos cuyo tamaño, complejidad y velocidad de crecimiento dificultan su captura, gestión, procesamiento o análisis mediante tecnologías y herramientas convencionales[8]. Esta definición nos lleva a la conclusión de que el análisis de tráfico de red es otro escenario más en el que nos encontramos con la problemática del *Big Data*.

La monitorización es una de las tareas más importantes en el análisis de tráfico, donde podemos encontrar dos métodos: el método tradicional de análisis de paquetes y el análisis de flujos.

El método tradicional proporciona una gran flexibilidad para la monitorización pero requiere una gran cantidad de recursos debido a la inspección profunda de paquetes, examinando e incluso exportando el contenido completo del paquete, presentando desafíos tanto con las tasas de datos como con el almacenamiento.

El segundo método, basado en flujos, procesa los paquetes exportando sólo los resúmenes del tráfico por flujo (agregación) reduciendo la cantidad de datos, pudiendo ser del orden de 1/2000 del volumen original, y aún así los conjuntos de datos pueden alcanzar las decenas de terabytes. Es por este motivo, por la reducción de los datos a procesar, y por su mayor escalabilidad, lo que hacen a este método la gran alternativa y solución a los problemas del método de análisis de paquetes individuales, llegando a ser el método elegido en el 90% de los anÁlisis de tráfico que se realizan, según Cisco [9][10].

Cabe destacar, que estos dos métodos no son excluyentes, siendo posible la utilización de los dos en conjunto.

1.3.1 Computación distribuida

La computación distribuida es un modelo de computación en paralelo, formado por un grupo de máquinas que son independientes entre sí y que pueden pertenecer a distintos dominios de administración de red. Utilizan estándares para poder llevar a cabo una tarea común, de forma que el usuario lo perciba como un único sistema.

Esta percepción de único sistema se debe a que los sistemas distribuidos comparten toda la carga entre las diferentes máquinas que lo forman para realizar las tareas de manera más eficiente y rápida gracias a juntar todos los recursos de red que forman todas las máquinas. Estos recursos principalmente son la CPU, la memoria y el almacenamiento. En la figura 1.1 podemos ver la comparación entre un sistema distribuido (a) y un único sistema (b).

La computación distribuida se puede implementar con diferentes arquitecturas, aunque normalmente la arquitectura de procesamiento por lotes, *batch*, suele ser la sinónima de *Big Data*. Estos lotes de datos se distribuyen a cada nodo del *cluster* permitiendo su procesamiento simultáneo, donde al aplicar el paradigma *MapReduce* se posibilita la creación de algoritmos distribuidos y escalables al implementar las clases *Map* y *Reduce*:

- *Map*: el nodo maestro toma la entrada y la divide en conjuntos de datos más pequeños, los cuales distribuye a los nodos trabajadores. El nodo trabajador los procesa y devuelve la respuesta al nodo maestro.

- *Reduce*: el nodo maestro recibe las respuestas y las combina para obtener el resultado final al problema que se trataba de resolver.

En la figura 1.2 podemos ver un breve diagrama de cómo sería el funcionamiento de *MapReduce* [11][12][13].

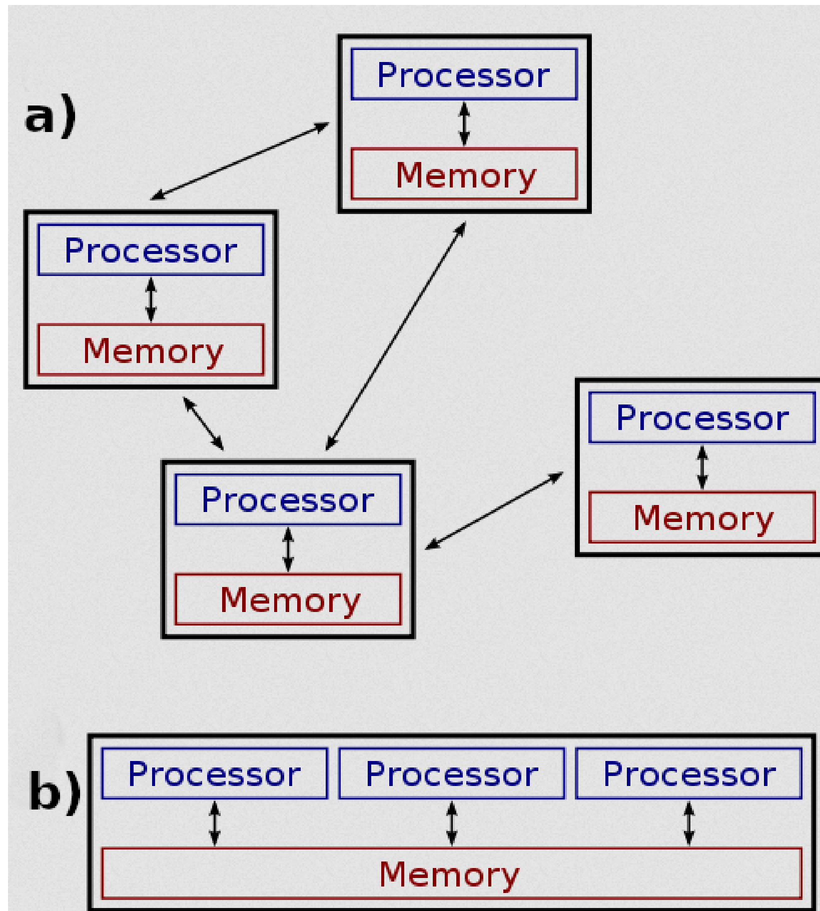


Figura 1.1: Diagrama de un sistema distribuido y un sistema en paralelo.

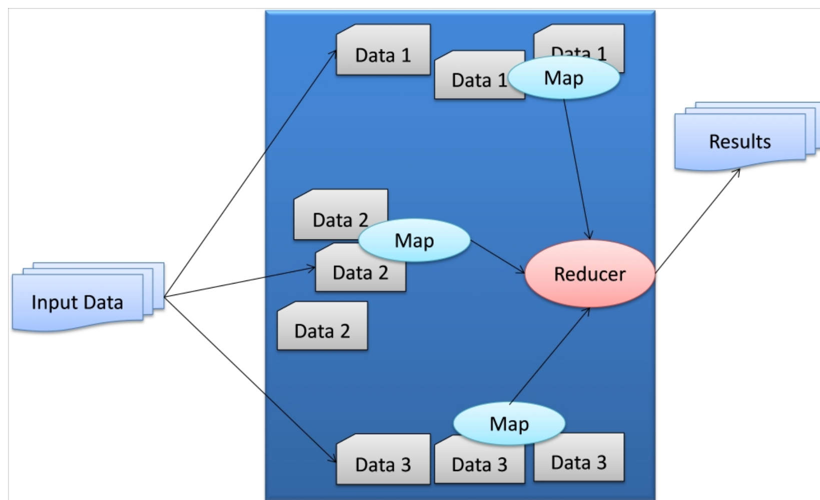


Figura 1.2: Breve diagrama del funcionamiento de *MapReduce*.

1.4 Objetivos

En este proyecto el objetivo principal es comparar y comprender diferentes técnicas de aprendizaje máquina empleadas en la clasificación de tráfico de red, marcando los siguientes objetivos a alcanzar:

- Estudio de distintas alternativas para el análisis de tráfico de red.
- Selección, implementación y comparación de diferentes algoritmos de aprendizaje máquina: se seleccionarán tres algoritmos para la tarea de clasificación.
- Aplicación de ingeniería de características sobre el conjunto de datos a estudio.
- Evaluación de los modelos resultantes.
- Configuración y despliegue de un sistema distribuido para ejecutar de forma paralela los modelos finales de *Machine Learning*.
- Evaluación de rendimiento del sistema distribuido: se realizarán pruebas de rendimiento en la plataforma distribuida, donde cada modelo final lo ejecutaremos con distintas configuraciones de nodos trabajadores, y posteriormente analizaremos la información resultante de la monitorización.

1.5 Repositorio Git

En el siguiente enlace se encuentra el repositorio con todo el código utilizado para la realización de este proyecto y también las gráficas sobre el rendimiento de las distintas pruebas del sistema distribuido: https://git.fic.udc.es/christian.varela.alvarez/TFG.git

Capítulo 2

Metodología

Para abordar un trabajo de minería de datos como este, consistiendo en descubrir patrones en grandes cantidades de datos utilizando técnicas de aprendizaje automático, existen varias metodologías, las cuales nos permiten estructurar en fases la realización del estudio para facilitar y agilizar el proceso de obtener conocimiento sobre los datos. Las tres metodologías más conocidas son: *Knowledge Discovery in Databases* (KDD), *Cross Industry Standard Process for Data Mining* (CRISP-DM) y *Sample, Explore, Modify, Model, Assess* (SEMMA) [14]. Estas metodologías comparten la esencia comentada, estructurando el proyecto en fases interrelacionadas entre sí, resultando en un proceso iterativo e interactivo [15].

Como diferencias entre estas metodologías tenemos que, la metodología SEMMA se centra más en las características técnicas del desarrollo del proceso, KDD se centra sobre todo en las tareas a realizar, y CRISP-DM se centra más en el propio proceso desde una perspectiva del proceso de negocio.

2.1 CRISP-DM

La metodología CRISP-DM fue concebida en 1996 bajo el programa de financiación ESPRIT de la Unión Europea, dando como resultado la primera versión de CRISP-DM en 1999.

Esta metodología consta de cuatro niveles de abstracción organizados jerárquicamente en tareas desde lo más general a lo más específico (figura 2.1). A nivel más general se describen las seis fases que componen el ciclo de vida de un proyecto de minería, donde cada etapa puede consistir en la realización de varias tareas. En la figura 2.2 podemos ver cómo se organiza el ciclo comentado.

Las fases se describen como:

- Comprensión del negocio: consiste en entender los objetivos y requerimientos del proyecto, así como definir el problema que nos abarca.

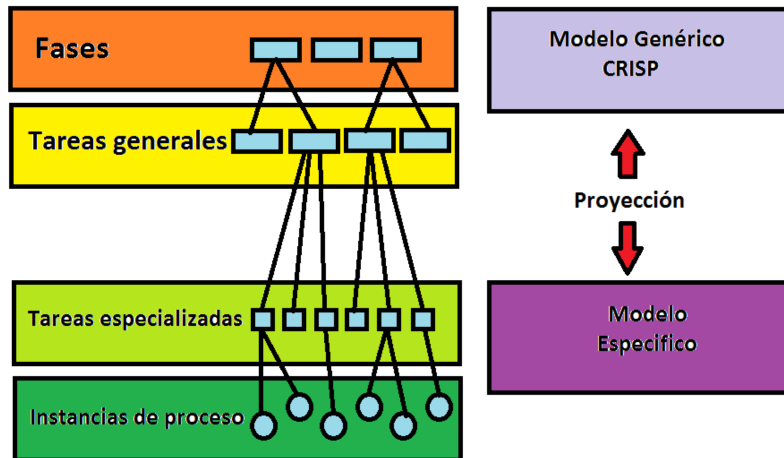


Figura 2.1: Los cuatro niveles de abstracción de CRISP-DM.

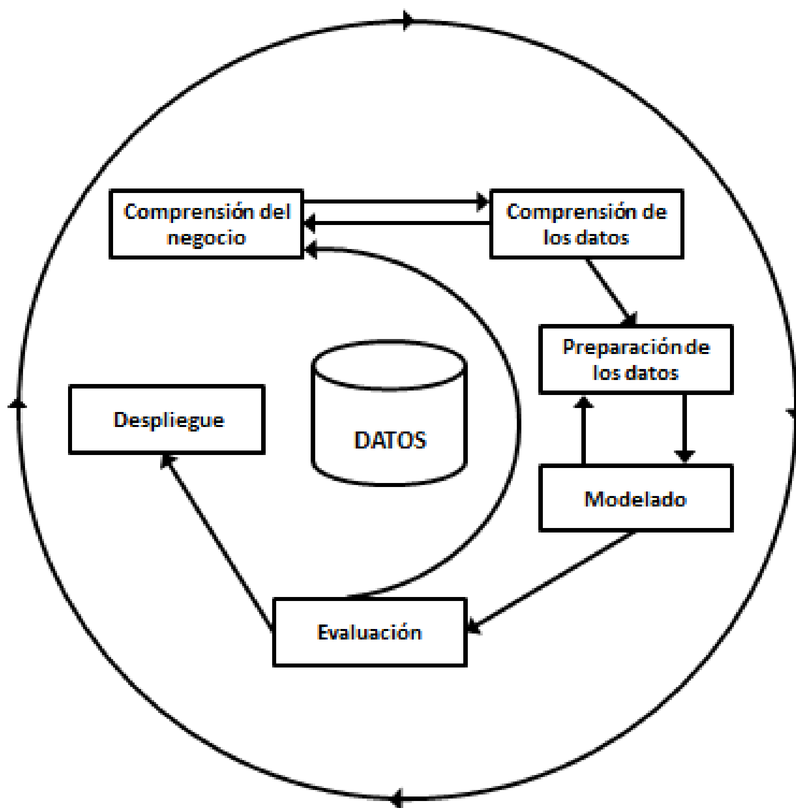


Figura 2.2: Las seis fases iterativas de la metodología CRISP-DM.

- Comprensión de los datos: consiste en la obtención y exploración del conjunto inicial de los datos, teniendo una primera toma de contacto con el problema visualizando la calidad de los datos y sacando unas hipótesis previas.

- Preparación de los datos: en esta etapa se realizan los pasos necesarios para posteriormente proceder al modelado, por lo que se realizan tareas como la selección de datos, limpieza, transformación y cambios de formato, y creación de nuevas variables, con el objetivo de obtener un conjunto de datos listo para ser modelado. Esta es la fase que requiere más tiempo y carga de trabajo.

- Modelado: en esta fase se seleccionan las técnicas de modelado más apropiadas para el trabajo que se está realizando, ajustando los parámetros necesarios para obtener unos resultados óptimos, y en caso de ser necesario se regresa a la etapa anterior para ajustar los datos correctamente. Finalmente se entrenan y validan los modelos creados.

- Evaluación: consiste en evaluar los modelos, teniendo en cuenta si se han cumplido los objetivos del negocio y revisar la integridad de todos los pasos, pudiendo repetir alguno en caso de que los resultados muestren algún posible error.

- Despliegue: se termina con la implantación de los modelos y mantenimiento, además de la obtención de conocimiento, por lo que hay que documentar y presentar los resultados de manera comprensible.

En la práctica, la secuencia de estas fases no es estricta y puede modificarse en función de la situación para facilitar la obtención de unos buenos resultados, por lo que estas seis etapas de CRISP-DM se tomarán como una guía, aunque en algún momento se siga un camino un tanto diferente.

A la hora de la redacción de la memoria, nos vamos a centrar más en la redacción del trabajo realizado en cada fase, y no tanto en el trabajo realizado en cada iteración, con el fin de facilitar su compresión.

Capítulo 3

Comprensión del negocio

En esta primera fase del proyecto nos tenemos que hacer una breve idea de lo que vamos a hacer, cómo y con qué, por lo que debemos de adquirir ciertos conocimientos teóricos y tecnológicos y explorar las herramientas que tenemos a nuestro alcance para intentar cumplir con los objetivos que nos hemos marcado, que se resumen en hacer clasificación de tráfico de red mediante técnicas de *Machine Learning*.

Las tareas de esta fase son las siguientes:

- Determinar los objetivos de negocio, tarea que ya hemos realizado cuando en la sección 1.4 definimos los objetivos marcados en el proyecto.
- Estudio del contexto en el que se realiza este proyecto, tarea que se realiza en las siguientes secciones.
- Estudiar e implementar la planificación a seguir en el proyecto. Esta tarea se realiza en la última sección de este capítulo, la sección 3.7.

En las sucesivas secciones haremos el estudio del contexto mencionado como segunda tarea, en la que primero explicaremos que es el ML y sus tipos de aprendizaje. Posteriormente explicaremos los algoritmos de clasificación escogidos y finalmente hablaremos del lenguaje de programación seleccionado, junto a las librerías y diferentes tecnologías que usaremos.

3.1 *Machine Learning*

El *Machine Learning* es un campo de investigación en el que se juntan la estadística, inteligencia artificial y la informática para extraer conocimiento de los datos creando sistemas que aprendan por sí solos sin la necesidad de intervención humana [16].

Este aprendizaje consiste en identificar patrones de una gran cantidad de datos aplicando un determinado algoritmo, dando como resultado la capacidad de predicción o toma de deci-

siones de manera automática basándose en el conocimiento que se adquirió sobre los datos conocidos.

Echando un vistazo más profundo sobre el ML, vemos que dispone de diferentes tareas a realizar, donde para cada una hay unos algoritmos concretos a aplicar, por lo que para elegir la tarea correcta tenemos que tener en cuenta cuál es nuestro objetivo y qué clase de información poseemos.

3.1.1 El objetivo principal

Como se ha comentado antes, ML nos ofrece las distintas tareas:

- Clasificación: las tareas de clasificación tienen como objetivo principal predecir a qué clase, de entre una lista ya predefinida, pertenece una muestra. Esta clasificación puede ser binaria, donde solo hay dos clases objetivo, o multiclase, en la que hay varias clases. Ejemplos de algoritmos: Regresión Logística, *Random Forest*, *Naive Bayes*, y *Support Vector Machine*.

- Regresión: las tareas de regresión tienen como objetivo principal predecir un número continuo. Ejemplos de algoritmos: Regresión Lineal, *Random Forest* y *Support Vector Regression*.

- *Clustering*: tarea que consiste en particionar el *dataset* en grupos llamados *clusters*. El objetivo es que dentro de un *cluster* las muestras sean muy parecidas entre ellas, y sean muy distintas al resto de muestras de los otros grupos. Ejemplos de algoritmos: k-Means, HCA…

- Visualización y reducción de la dimensionalidad: tarea que tiene como objetivo reducir el tamaño de los datos eliminando aquellas características que no son relevantes. Esta necesidad de reducción se puede deber a tener que usar un método que no puede procesar tantas características como tenemos, o ser imposible hacer gráficas con una cantidad tan grande datos. Ejemplos de algoritmos: PCA, LLE, t-SNE.

- Reglas de asociación: tarea que permite descubrir las asociaciones entre los elementos de un conjunto de datos relevantes. Ejemplos de algoritmos: Apriori, Eclat [17].

3.1.2 Tipo de aprendizaje

Además de las tareas comentadas en el apartado anterior, debemos tener en cuenta cómo es nuestro conjunto de datos, puesto que en función de eso debemos elegir un tipo de aprendizaje u otro. Existen dos tipos de aprendizaje[18]:

- Aprendizaje supervisado: el usuario proporciona al algoritmo unos datos de entrada y la salida deseada, y el algoritmo debe encontrar la forma de producir la salida esperada según la entrada proporcionada. Para darse este caso, el conjunto de datos debe estar bien etiquetado.
  A modo de resumen, se puede decir que este tipo de aprendizaje es supervisado debido a que de antemano ya sabemos la salida que debe producir. Se suele usar en tareas de clasificación y de regresión.

- Aprendizaje no supervisado: en este caso el usuario proporciona al algoritmo solo los datos de entrada, puesto que los de salida se desconocen al estar sin etiquetar el conjunto de datos, por lo que solo podemos tener una breve idea de la estructura de los datos. Es por esto por lo que se suele utilizar en tareas de *clustering*, entre otras.

3.2 Algoritmos

A la hora de elegir los tres algoritmos marcamos un requisito principal: intentar escoger algoritmos que fuesen de una naturaleza lo más distinta posible. Por otro lado también tenemos una condición impuesta por la librería que vamos a usar para la parte de distribución puesto que los algoritmos escogidos deben estar implementados tanto en la librería Scikit-Learn como en MLlib. Es una condición bastante importante, puesto que la primera librería posee una gran cantidad de algoritmos, mientras que la segunda es más modesta en ese aspecto, por lo que de esta forma llegado el momento de distribuir nos evitaremos encontrarnos con la situación de no poder utilizar los mismos algoritmos.

Los algoritmos implementados en MLlib los podemos ver en la figura 3.1.

Problem Type	Supported Methods
Binary Classification	linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes
Multiclass Classification	logistic regression, decision trees, random forests, naive Bayes
Regression	linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression

Figura 3.1: Algoritmos implementados en MLlib [1].

En un primer momento podemos hacer uso de alguna plantilla como la de la figura 3.2 para tener una breve idea de qué algoritmos poder seleccionar, teniendo en cuenta que nuestro trabajo consiste en una tarea de clasificación. Tras hacer una búsqueda por la web comparando los resultados de los distintos algoritmos escogemos los tres siguientes:

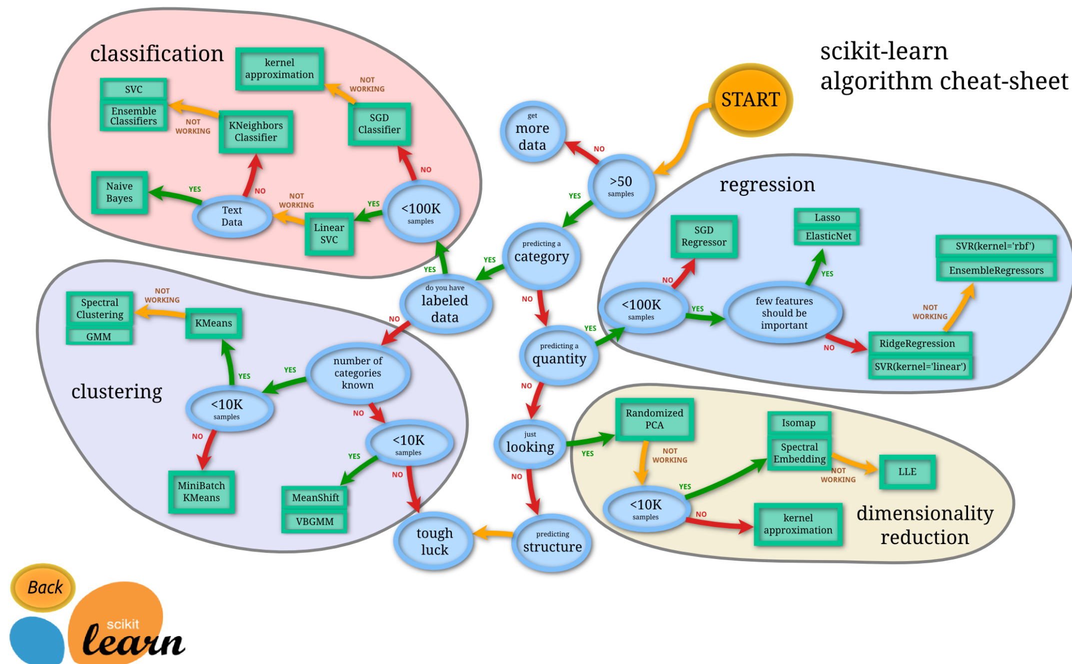


Figura 3.2: Plantilla de Scikit-Learn para la elección de los algoritmos.

- Regresión Logística: algoritmo muy recomendado para probarlo de primero y tener una breve idea de cómo funciona el ML al devolver unos resultados fáciles de entender. Buen rendimiento con grandes conjuntos de datos.

- *Random Forest*: algoritmo potente y robusto, gestionando muy bien el ruido debido a su naturaleza.

- *Naive Bayes*: algoritmo muy rápido incluso con grandes cantidades de datos. Su naturaleza es muy distinta a la de los anteriores, pero también la cantidad de información sobre su uso es mucho menor, por lo que en parte también se escoge por este motivo, marcando como reto su utilización a pesar de este inconveniente.

3.2.1 Regresión Logística

El nombre de este algoritmo a veces puede llevar a confusión al contener la palabra regresión, puesto que el propósito de la regresión es establecer un modelo para relacionar un cierto número de características con una variable objetivo continua. Esto es lo que se conoce como Regresión Lineal, debido a que su propósito es establecer una línea arbitraria y calcular la distancia de la recta a los puntos de datos [19].

Esta breve teoría se puede ajustar para utilizar algunos algoritmos de regresión para la clasificación, dando lugar a la Regresión Logística, que consiste en estimar la probabilidad de

que una instancia pertenezca a una clase particular. Si la probabilidad estimada es superior al 50% el modelo predice que la muestra pertenece a una clase, llamémosla Clase 1, mientras que en los otros casos predice que pertenece a la otra clase, Clase 0. Por lo tanto estaríamos hablando de un algoritmo de clasificación binaria, que es la tarea de este trabajo.

Su funcionamiento es muy parecido al de la Regresión Lineal, calculando la suma ponderada de las características de entrada, pero se diferencia al no mostrar esta suma como resultado, sino que genera la logística de esta suma. Esta logística, también llamada *logit* es una función sigmoidea cuya salida es un número entre cero y uno. La definición de esta función y su gráfica las podemos ver en la figura 3.3.

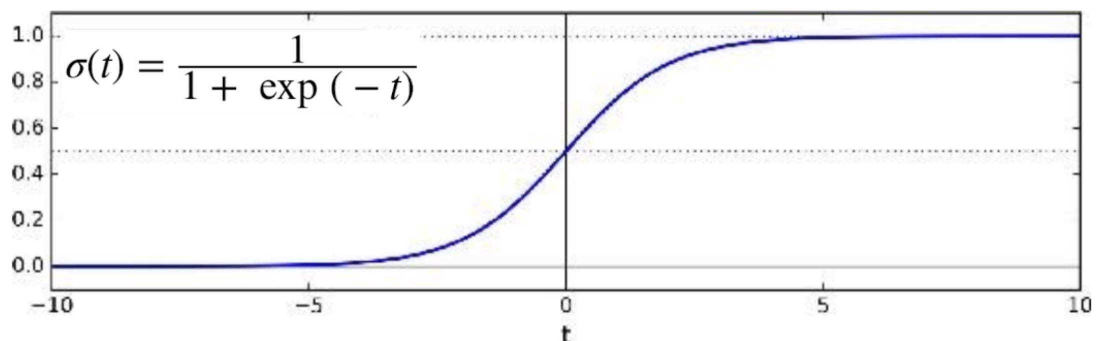


Figura 3.3: Fórmula de la función sigmoidea y su gráfica.

3.2.2 *Random Forest*

Random Forest es unos de los algoritmos más utilizados en *Machine Learning* debido a sus buenos resultados. Pero para explicar este algoritmo, primero es importante tener una idea de cómo funciona el algoritmo de Árboles de Decisión.

Árboles de Decisión es un algoritmo ampliamente usado para tareas de clasificación y regresión, donde básicamente cada árbol es una jerarquía de preguntas if/else que dan lugar a una decisión.

Por lo tanto, los árboles tienen un primer nodo llamado raíz y luego el resto de atributos de entrada se descomponen en dos ramas (o más) planteando una condición que puede ser cierta o falsa. De esta forma, cada nodo se bifurca en dos y vuelven a subdividirse hasta llegar a las hojas, los nodos finales, que contienen las respuestas. La estructura de un Árbol de Decisión lo podemos ver en la figura 3.4.

Para obtener el árbol óptimo, el algoritmo mide las predicciones logradas y las compara haciendo uso de unas funciones, siendo las más usadas el ”Índice Gini” y la ”Ganancia de la información” [20].

Las ventajas de este algoritmo son que es rápido y muy simple de entender, puesto que

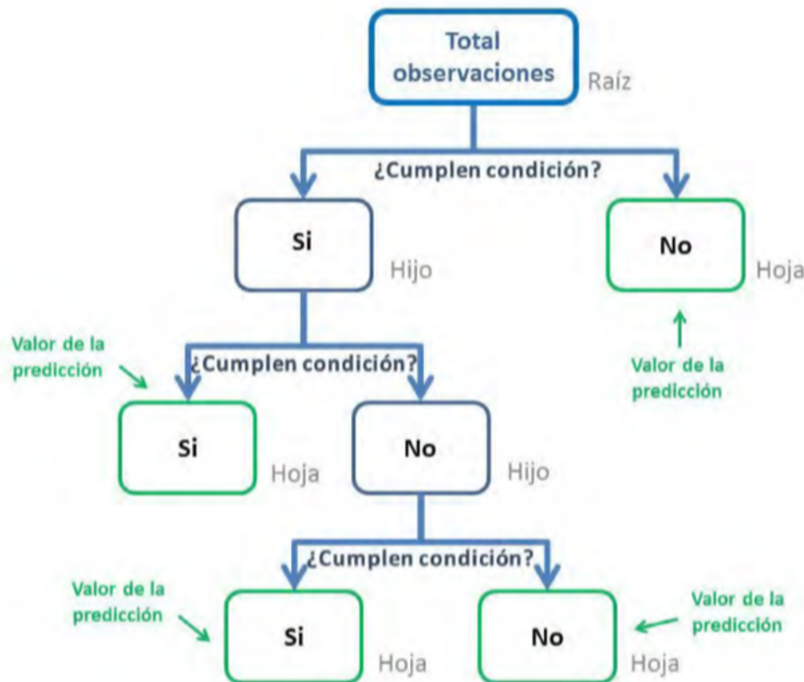


Figura 3.4: Ejemplo de Árbol de Decisión [2].

los resultados son interpretables gráficamente, y los datos no requieren estar escalados. Una de las grandes desventajas que presenta es que el algoritmo tiende a sobreajustar los datos de entrenamiento.

Random Forest es un algoritmo que combina las ventajas del algoritmo Árboles de Decisión e intenta solventar sus desventajas, creando una colección de árboles de decisión, donde cada árbol es ligeramente diferente de los demás. De esta forma tenemos muchos árboles que hacen unas buenas predicciones y se sobreajustan de diferentes formas, por lo que podemos reducir el sobreajuste promediando los resultados. Para asegurar esta reducción de sobreajuste tenemos que garantizar que los árboles sean diferentes, y es aquí donde cobra sentido el nombre del algoritmo: bosque aleatorio, puesto que el algoritmo inyecta aleatoriedad en la construcción de los árboles.

Los árboles que forman el bosque serán construidos independientemente uno del otro y el algoritmo tomará diferentes decisiones aleatorias para asegurar que los árboles sean distintos. Para comenzar la construcción de un árbol, primero se toma una muestra de arranque de nuestros datos. Esta muestra será tan grande como el conjunto de datos original, salvo que algunos datos faltarán (aproximadamente un tercio) y otros estarán repetidos. Esta técnica se denomina *Bootstrap*, y proporciona cierta aleatoriedad al tener conjuntos de entrada diferentes para los árboles.
El siguiente paso es crear el árbol de decisión a partir de esta muestra, salvo que a diferencia

del algoritmo Árboles de Decisión, *Random Forest* no busca la mejor prueba para cada nodo, sino que selecciona aleatoriamente un subconjunto de las características, y busca entre ellas la mejor prueba.

Estos dos pasos que acabamos de describir son los que garantizan que todos los árboles en el bosque sean diferentes. Una vez creado el bosque aleatorio, para predecir, el algoritmo hace una predicción por cada árbol, y se promedian las probabilidades predichas para cada clase, dando como resultado la clase con mayor probabilidad.

Random Forest tiene como ventaja su gran potencial, dando muy buenos resultados sin tener que realizar un gran ajuste en los parámetros, no requiere que los datos estén escalados, y reduce el sobreajuste.

Como desventajas podemos destacar el alto coste computacional si se trabaja con un alto volumen de datos, y también la difícil interpretación del modelo (en comparación con los árboles de decisión).

3.2.3 *Naive Bayes*

Naive Bayes es un algoritmo de clasificación muy parecido a los algoritmos lineales, aunque tiende a ser aún más rápido a costa de empeorar su rendimiento en la generalización.

Este algoritmo se basa en el teorema de Bayes, cuyo funcionamiento, a muy grandes rasgos, lo podemos definir como la probabilidad de que ocurra A dado que ocurrió B, siendo B la evidencia y A la hipótesis. En la figura 3.5 podemos ver la fórmula del teorema. El algoritmo

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Figura 3.5: Fórmula del teorema de Bayes [3].

asume que las características son independientes entre ellas, donde la presencia de una característica no afecta a otra, lo cual simplifica bastante la computación. Es por este motivo por lo que se llama *Naive* (ingenuo).

Hay tres tipos de clasificadores *Naive Bayes*:

- Multinomial: usado frecuentemente para la clasificación de documentos. Las características usadas por el clasificador son la frecuencia con la que aparecen las palabras en el documento.

- Bernoulli: es similar al multinomial, pero los valores de las características son ceros o unos.

- Gaussian: cuando las características toman valores continuos y no son discretos, se asume que los valores están muestreados con una distribución gaussiana.

Como ventajas podemos destacar: es fácil y rápido predecir la clase de conjunto de datos de prueba, cuando se mantiene la suposición de independencia el algoritmo funciona incluso mejor que los modelos lineales y necesita menos datos de entrenamiento, funciona bien en el caso de variables de entrada categóricas comparada con variables numéricas.

Como desventajas nos encontramos con que los algoritmos de *Naive Bayes* son conocidos por ser pobres estimadores, la presunción de independencia *Naive* no se suele reflejar en los datos del mundo real, y si el conjunto de datos de prueba tiene una característica que no ha sido observada en el conjunto de entrenamiento, el modelo le asignará una probabilidad de cero y será imposible realizar predicciones [21][22].

3.3 Python y PySpark

Python es uno de los lenguajes de programación más populares, y se está convirtiendo en una de las mejores opciones para el desarrollo de proyectos de inteligencia artificial, *Machine Learning* y *Deep Learning*.

Esta popularidad se debe a la amplia selección de librerías y *frameworks* que posee, donde con relación al objeto de este proyecto, las más comunes pueden ser Pandas, Scikit-Learn o Numpy. Este lenguaje además destaca por su código conciso, legible y por la agilidad que ofrece su sintaxis simple [23].

Otra de las ventajas de usar Python es su fuerte similitud con PySpark y su posterior uso para la etapa de distribución.

PySpark es un API de Python para usar en Apache Spark, facilitando el desarrollo de programas eficientes gracias a las librerías que permite utilizar, como pueden ser PySparkSQL o MLlib. Entre las grandes ventajas de PySpark está el soporte de uso con otros lenguajes como Scala, Java y R, facilidad de trabajo con RDDs (*Resilient Distributed Datasets*) y su gran rapidez en comparación con otros *frameworks* de procesamiento de datos [24][25].

3.4 Scikit-Learn

Scikit-Learn posiblemente sea la librería más utilizada para ML en Python, puesto que es de código abierto, y proporciona una gran gama de algoritmos de aprendizaje supervisados y no supervisados.

Otro aspecto muy importante es que permite cubrir todas las fases de un proyecto de ML, como por ejemplo al tener disponible varios métodos para la validación cruzada.

Su elección se basó en estas ventajas mencionadas, además de por su gran cantidad de documentación en su página web, y por la gran comunidad que posee, donde no solo hay una inmensa cantidad de foros y libros en los que se hable y utilice esta librería, sino que también está en continua mejora al tener unos treinta y cinco colaboradores desarrollándola [26].

3.5 Apache Spark y Databricks

Apache Spark es un *framework Open Source* de programación para el procesamiento de datos distribuidos diseñado para que sea rápido y de propósito general. Además, el gran número de APIs y módulos que incluye permiten una gran flexibilidad e interconexión con otras herramientas de distribución como pueden ser Hadoop, Hive o Kafka. Estas cualidades hacen de Spark una herramienta muy importante para su uso en el mundo del *Big Data* y del aprendizaje automático al proporcionar una gran potencia para procesar grandes conjuntos de datos.

Su arquitectura la podemos ver en la figura 3.6, formada por los siguientes componentes:

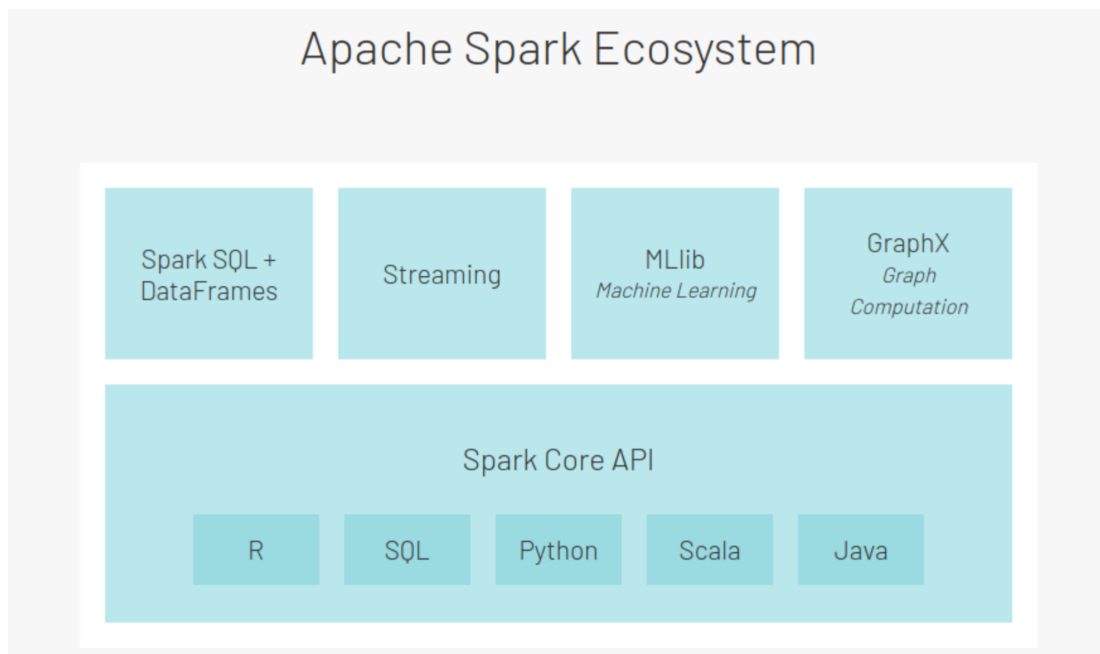


Figura 3.6: Arquitectura de Apache Spark [4].

- SparkSQL: es el módulo para el procesamiento de datos estructurados y semi-estructurados. Con este módulo vamos a poder transformar y realizar operaciones sobre los RDD o los *dataframes*. Está pensando exclusivamente para el tratamiento de los datos.
- Streaming: habilita la capacidad de procesar y analizar flujos de datos nuevos en tiempo

real. Hereda las características de tolerancia a fallos y facilidad de uso de Spark. Se integra fácilmente con un alto número de fuentes de datos.

- MLlib: librería muy completa y escalable que contiene numerosos algoritmos de ML, tanto de *clustering*, como de clasificación y regresión.

- GraphX: motor de cálculo de gráficos que permite construir, transformar y razonar interactivamente sobre los gráficos de datos estructurados.

- Spark Core: motor de ejecución general de la plataforma sobre el que se construyen las demás funcionalidades.

Las bondades de Spark, además de las ya mencionadas, son su escalabilidad lineal, la tolerancia a fallos y las funcionalidades DAG y RDD.

Directed Acyclic Graph (DAG) es un grafo dirigido que no tiene ciclos. Spark soporta flujos de datos acíclicos, donde cada tarea de Spark crea un DAG de etapas de trabajo para que se ejecuten en el *cluster*. Esta característica hace que Spark sea mucho más rápido que Hadoop (su predecesor con el que se le suele comparar), puesto que Spark no tiene que escribir en disco los resultados obtenidos en las etapas intermedias del grafo, mientras que Hadoop sí que tiene que hacerlo.

Por otro lado están los *Resilient Distributed Datasets* (RDD), que permiten realizar operaciones sobre grandes cantidades de datos de una manera rápida y tolerante a fallos.

Cuando los datos ya han sido leídos como objetos RDD, estos se dividen en particiones lógicas, permitiendo la paralelización. Se pueden realizar transformaciones (se obtiene un nuevo RDD) o realizar una acción (se aplica una operación y se obtiene un valor como resultado), donde cada máquina realiza una parte de la tarea.

En cuanto a la instalación de Apache Spark, se puede realizar en modo *Standalone*, donde en cada nodo del *cluster* se instala la misma instancia (figura 3.7).

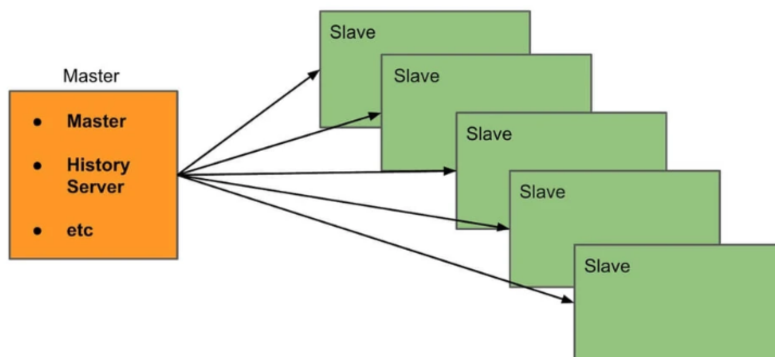


Figura 3.7: Instalación de Apache Spark en modo *Standalone* [5].

También se puede instalar usando un gestor de recursos, el cual se encarga de repartir las tareas entre las distintas máquinas del *cluster*, y también tenemos el *driver* o *main*, el cual se encarga de levantar el SparkContext, que es el que coordina que las aplicaciones se ejecuten como un grupo independiente de procesos en el *cluster*. En la figura 3.8 podemos ver cómo sería esta instalación.

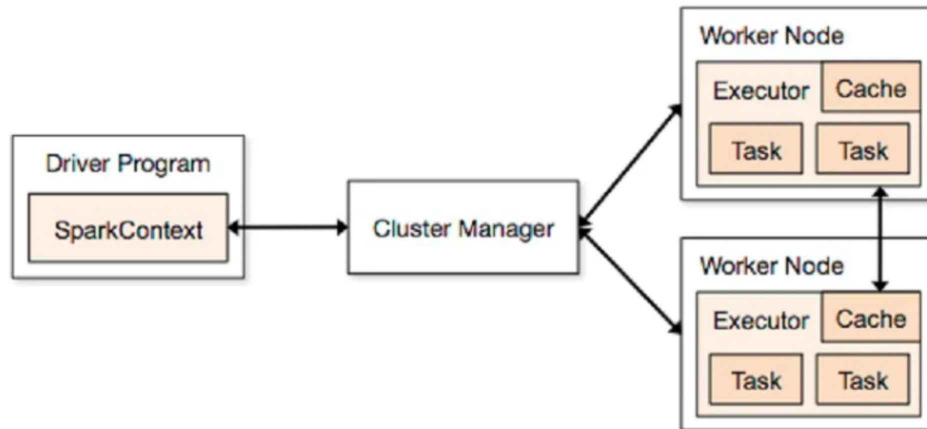


Figura 3.8: Instalación de Apache Spark con gestor de recursos [5].

En cuanto a Databricks, es una plataforma analítica de datos que se basa en Apache Spark, siendo ideal para hacer analítica *Big Data* e inteligencia artificial de una forma bastante sencilla y colaborativa gracias a sus *notebooks*.

También destaca la facilidad con la que se permite auto-escalar y dimensionar los entornos de Spark, facilitando también el despliegue de los *clusters*, resultando muy cómoda la configuración del entorno (ver figura 3.9).

Esta plataforma está disponible como servicio *cloud* tanto en Microsoft Azure como en Amazon Web Services (AWS), de los cuales utiliza su infraestructura, como por ejemplo para el almacenamiento, y una de las ventajas es que integra Ganglia, una herramienta de monitorización, tanto del *cluster* como de los nodos por separado. También permite la carga de ficheros mediante la interfaz gráfica, lo cual resulta muy útil.

La capacidad y herramientas de las que dispone de forma nativa para la creación de gráficos y su visualización hacen, que junto a lo comentado anteriormente, resulte una plataforma muy completa para desarrollar todas las fases de un proyecto de *Machine Lerarning* [27][28].

3.6 Apache Kafka

En un mundo "siempre conectado" la transmisión de datos se está convirtiendo en la base fundamental de la que partir. Esta transmisión de datos consiste en capturar datos a partir de

Figura 3.9: Pantalla de configuración de un *cluster* en Databricks.

eventos de distintas fuentes, almacenarlos de forma duradera, manipularlos y procesarlos, y enviar estos flujos de datos a los diferentes destinos que pueden ser de diversas tecnologías, todo ello en tiempo real.

Apache Kafka es una plataforma distribuida de transmisión de datos (*event streaming*) que permite publicar, almacenar y procesar flujos de datos en tiempo real. Las ventajas de estas funcionalidades es que se proporcionan de manera distribuida, escalable, elástica, tolerante a fallos y de forma segura.

Este sistema distribuido consiste en servidores y clientes que se comunican en la red a través del protocolo TCP. Los servidores Kafka se ejecutan como un *cluster*, donde algunos de estos servidores forman la capa de almacenamiento, mientras que otros ejecutan Kafka Connect para que de forma continua poder importar y exportar los datos como flujos de eventos y así integrar Kafka con los demás sistemas.

Por otro lado, los clientes son aplicaciones distribuidas y microservicios que leen, escriben y procesan flujos de eventos de forma paralela. Concretamente las aplicaciones que publican

los eventos hacia Kafka se denominan Productores, mientras que las aplicaciones que están suscritas a los eventos, esto es, que leen y procesan los datos, se denominan Consumidores.

En la figura 3.10 podemos ver cómo se relacionan los clientes con Kafka. En esta figura aparece un término también muy importante, los *topics*, que son la forma de organizar y almacenar los eventos.

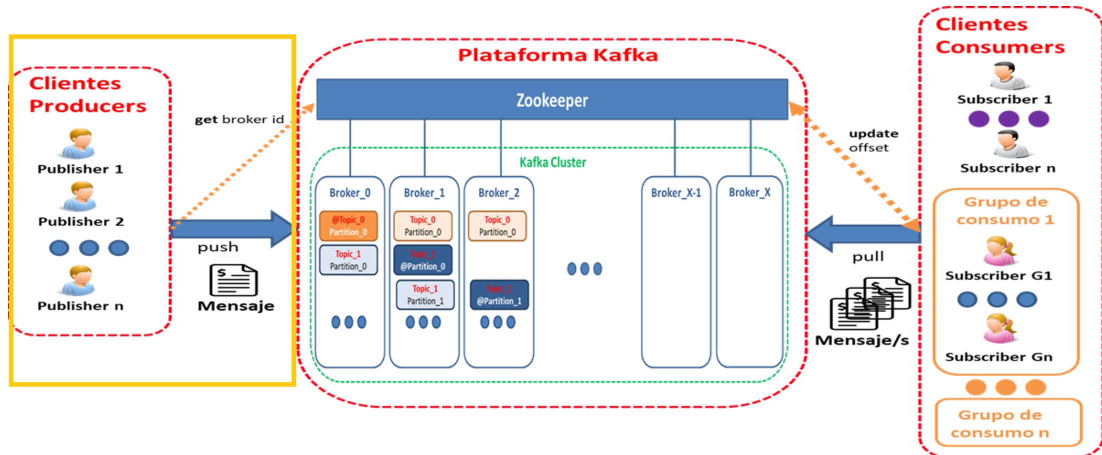


Figura 3.10: Diagrama del funcionamiento de Kafka.

Estos temas (*topics*) están divididos en particiones que se almacenan de forma distribuida para proporcionar escalabilidad.

Apache Kafka proporciona varias APIs en varios lenguajes de programación para poder implementar toda la funcionalidad que acabamos de describir [29][30].

3.7 Planificación

En este proyecto se ha utilizado Scrum como metodología para la planificación, mientras que para las tareas que lo componen hemos utilizado CRISP-DM (sección 2.1).

Scrum es un marco de trabajo con el cual las personas pueden abordar problemas complejos y crear productos con el máximo valor de forma productiva y creativa.

Este marco de trabajo se engloba dentro de las metodologías ágiles, puesto que se basa en aspectos como:

- Flexibilidad ante la aparición de cambios y nuevos requisitos durante la elaboración del proyecto.
- El factor humano.
- La colaboración e interacción con el cliente.

- El desarrollo iterativo para asegurar buenos resultados.

Scrum se basa en la teoría empírica, en la que el conocimiento procede de la experiencia y de tomar decisiones apoyándose en lo que ya se conoce. El control de procesos empírico se fundamenta en tres claves:

- Transparencia: todos los implicados tienen conocimiento del estado del proyecto. Esto ayuda a tener un mayor entendimiento común.

- Inspección: los miembros del equipo Scrum inspeccionan frecuentemente el progreso del trabajo para detectar posibles problemas.

- Adaptación: el equipo se ajusta para conseguir el objetivo del *sprint* ante la aparición de cambios.

Los equipos de Scrum para conseguir entregar valor y ofrecer resultados de calidad tienen que ser auto-organizados y multifuncionales. Dentro de los equipos existen tres roles:

- *Product Owner* (dueño del producto): es el responsable de obtener el mayor valor posible del trabajo del equipo de desarrollo. Debe tener un amplio conocimiento del negocio, puesto que es la persona que habla constantemente con el cliente.

- *Scrum Master*: es el responsable de que las técnicas Scrum se comprendan y apliquen en el proyecto. Debe ayudar en la adopción de la metodología Scrum en todos los equipos.

- Equipo de desarrollo: equipo multifuncional y auto-organizado que realiza las tareas priorizadas por el dueño del producto para desarrollar el producto.

En Scrum existen eventos predefinidos que tienen como fin crear regularidad y minimizar la necesidad de salirse de lo pautado por Scrum. Los eventos son:

- *Sprint*: es el corazón de esta metodología, ya que contiene los demás hitos del proceso. Todo lo que ocurre dentro de una iteración para crear valor está dentro de un *sprint*. La duración máxima debe ser de un mes.

- *Sprint Planning* (Planificación de *Sprint*): es una reunión en la que todo el equipo de Scrum define las tareas a abordar y el objetivo del *sprint*.

- *Daily Scrum* (Scrum Diario): es una reunión diaria en la que deben participar el equipo de desarrollo y el *Scrum Master*, y en ella se habla de lo que se hizo el día anterior, de lo que se va a hacer hoy y si hay algún impedimento que se necesita que lo solucionen. Suele tener una duración corta.

- *Sprint Review* (Revisión de *Sprint*): reunión a la que asiste el cliente y el *Product Owner* le presenta lo desarrollado, mientras que el equipo de desarrollo le muestra su funcionamiento. El cliente valida los cambios realizados y ofrece *feedback* sobre posibles nuevas tareas.
- *Sprint Retrospective* (Retrospectiva de *Sprint*): último evento de Scrum, que consiste en una reunión en la que el equipo evalúa cómo se ha implementado la metodología Scrum en el último *sprint*. Tiene como resultado una lista de mejoras que serán abordadas durante el siguiente *sprint*.

Para maximizar la transparencia también se definen unas herramientas o artefactos, que son los siguientes:

- *Product Backlog* (Lista de Producto): es un listado ordenado de tareas que engloba todo un proyecto. Todo lo que se deba hacer debe estar en este listado junto a su tiempo estimado.
- *Sprint Backlog* (Lista de Pendientes del Sprint): grupo de tareas que el equipo de desarrollo elige en la planificación del *sprint* junto con el plan para desarrollarlas. Debe ser conocido por todo el equipo.
- Incremento: es la suma de todos los elementos de la lista de producto completados durante un *sprint* y el valor de los incrementos de todos los *sprint* anteriores. Al final de un *sprint* el nuevo incremento debe estar en condiciones de poder ser utilizado.

Para la consecución de los incrementos se deben de realizar las historias de los usuarios. Las historias son breves requisitos o solicitudes escritas y ordenadas desde el punto de vista del propietario del producto (estos requisitos se encuentran en la lista de producto que acabamos de comentar).

Cada historia debe describir las tareas o subtareas en las que se descompone, decidiendo qué pasos deben completarse y quién es el responsable, facilitando así la gestión y el desarrollo [31][32].

Un conjunto de historias forman el desglose de grandes cantidades de trabajo, que se denominan épicas, y el conjunto de épicas forman las iniciativas, que conducen hacia un objetivo común. Las épicas es algo que se puede completar en uno o varios meses, mientras que las iniciativas suelen completarse en varios trimestres. La figura 3.11 puede ayudar a entender mejor la estructura que siguen las iniciativas, épicas e historias [33].

Dicho esto, como se utiliza CRISP-DM como metodología a seguir para las tareas, el flujo marcado por CRISP-DM marcará las tareas a implementar en cada historia de usuario. Tendremos un *sprint* por cada historia de usuario, donde la lista de tareas que las componen, junto a la duración estimada lo podemos ver en la tabla 3.1.

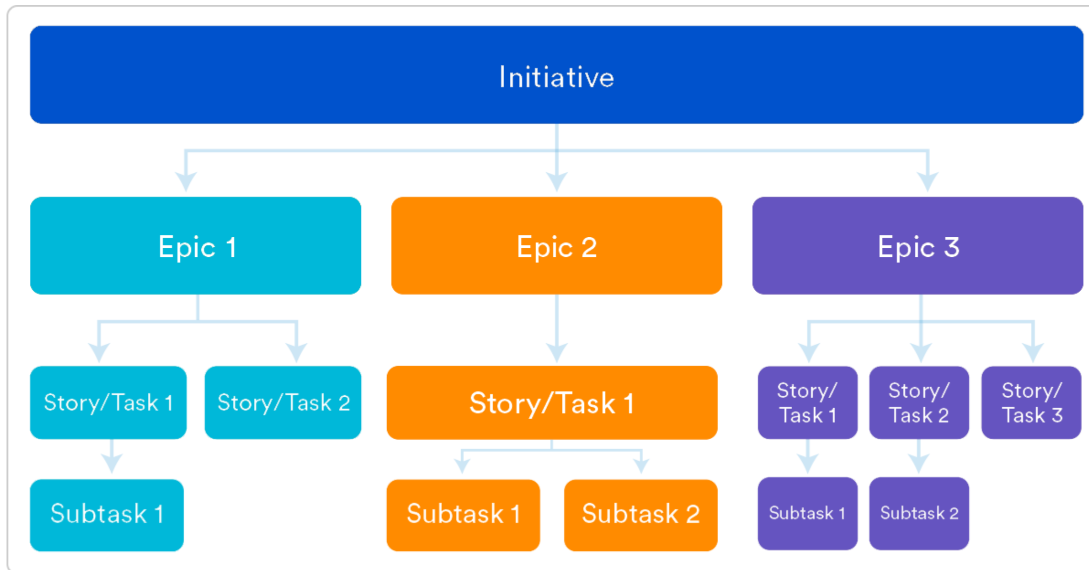


Figura 3.11: Estructura de las iniciativas en la gestión ágil de proyectos.

La estimación de la duración del proyecto es de unas 300 horas. En cuanto al coste, nos guiamos por el BOE [34] para calcular el precio del tiempo dedicado de los recursos humanos, donde mi puesto profesional sería el clasificado como el nivel II del grupo B del área de Consultoría, Desarrollo y sistemas, mientras que los dos directores pertenecerían al grupo A de la misma área. Según esto, el coste por hora sería de 16,9 y 18,2 euros respectivamente, incluyendo ya otros pagos como el seguro de la seguridad social. Por lo tanto, mi trabajo supondría un coste de 5.070 euros y el de cada director unos 254,80 euros por las catorce horas estimadas de su trabajo.

En cuanto al coste de los recursos utilizados para realizar el despliegue de los sistemas distribuidos, suponiendo el uso de Databricks y de Apache Kafka utilizando la infraestructura de Microsoft Azure durante unas cincuenta horas, el importe ascendería a 22,3 euros, desglosado de la siguiente forma:

- Azure Databricks utilizándolo para una carga de trabajo de análisis de datos: 0,34 euros/hora (a parte de otros costes como la IP pública, por ejemplo), según su página web.

- Apache Kafka en Azure HDInsight, utilizando nodos A2v2 de uso general: 0,106 euros/hora (a parte de otros posibles costes), según su página web.

Por lo tanto, el coste estimado del proyecto es de 5.602 euros.

Historias de usuario	Lista de Tareas	Estimación (horas)
Entender el negocio	Estudio de los fundamentos teóricos sobre el aprendizaje automático	25
	Estudio de los fundamentos teóricos y tecnológicos sobre los flujos de red	10
	Estudio de los fundamentos teóricos y tecnológicos sobre los sistemas distribuidos	16
	Total	51
Modelo de regresión logística	Estudio del algoritmo de Regresión Logística	15
	Entender los datos	14
	Preprar los datos	15
	Modelar	8
	Evaluar	6
	Total	56
Modelo de Random Forest	Estudio del algoritmo de Random Forest	25
	Preparar los datos	10
	Modelar	10
	Evaluar	6
	Total	51
Modelo de Naive Bayes	Estudio del algoritmo de Naive Bayes	12
	Preparar los datos	14
	Modelar	8
	Evaluar	6
	Total	40
Despliegue	Configuración del sistema distribuido	30
	Pruebas de rendimiento	20
	Total	50
Redacción de la memoria	Introducción	6
	Planificación del proyecto	2
	Metodología	2
	Entender el negocio	6
	Entender los datos	4
	Preprar los datos	8
	Modelar	6
	Evaluar	4
	Desplegar	10
	Líneas futuras y conclusiones	4
	Total	52

Tabla 3.1: Planificación del proyecto.

Capítulo 4

Comprensión de los datos

4.1 El *dataset*

Para la realización de un trabajo como este se necesita un conjunto de datos lo más real posible, lo cual a veces puede ser difícil de obtener debido a problemas de privacidad, ya sea porque son datos internos de una organización y no se comparten, o sí se comparten pero mostrando muy poca información, llegando a tal punto de no ser útiles para el estudio. Para paliar este problema la Universidad de New Brunswik creó un escenario de pruebas para simular tráfico de red realista, dando como resultado capturas de tráfico en archivos de tipo *pcap* y el *dataset* que se va a utilizar en este trabajo, llamado UNB ISCX IDS 2012 [35].

Este conjunto de datos tiene la ventaja de proporcionar información muy parecida a la que se podría encontrar en una red real y de estar ya etiquetado, lo que permite no tener que analizar los archivos *.pcap* para crear las etiquetas manualmente. Esta característica facilita la realización de ingeniería de características más adelante. Además, el alto número de muestras que contiene nos será útil para crear un conjunto lo suficientemente bueno para el entrenamiento y otro para las pruebas.

Podemos observar en la figura 4.1 que el escenario simula una red real, el cual tiene tráfico de red normal y también varios casos de tráfico de ataque, que son los siguientes:

- Intento de intrusión desde dentro de la red
- HTTP DoS
- Ataque DoS utilizando una IRC Botnet
- Ataque de fuerza bruta sobre SSH

Algo a destacar es que los tipos de ataques están separados por días. Esto quiere decir que en cada día solo encontraremos tráfico normal o tráfico de un tipo de ataque. Esto puede

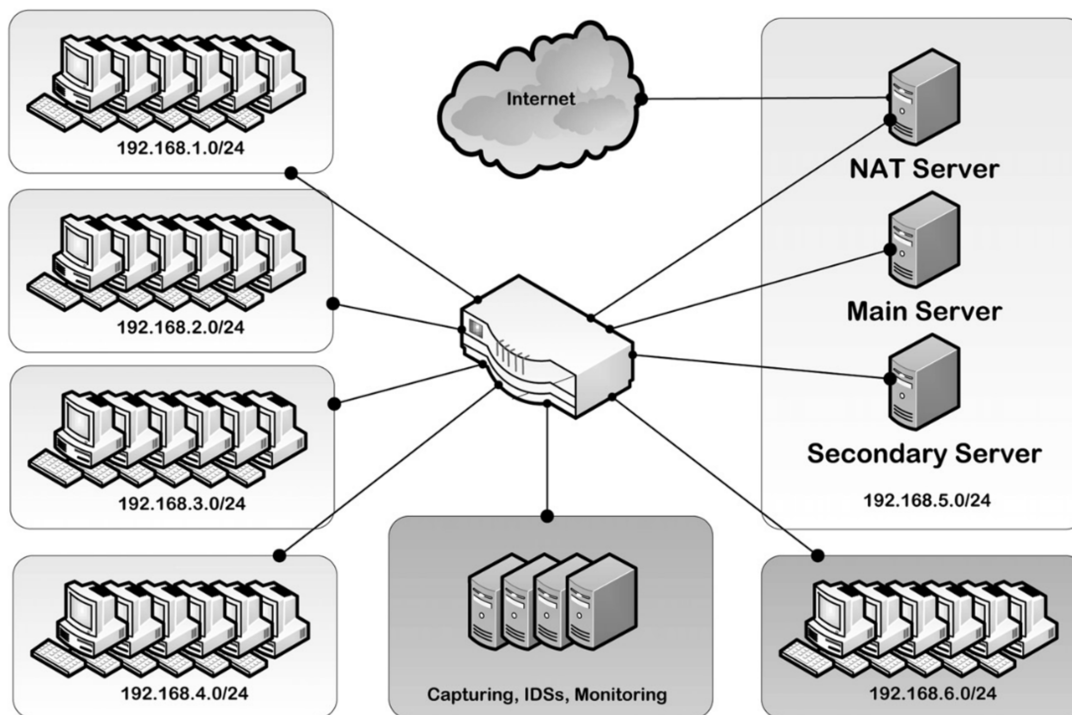


Figura 4.1: Diagrama del escenario del *dataset* UNB ISCX IDS 2012.

suponer una pequeña desventaja si por motivos de rendimiento decidiésemos utilizar los datos de un único día para realizar las tareas del proyecto.

4.2 Importación de los datos

Como se comentó en la sección anterior, usaremos un *dataset* ya creado, evitando el trabajo de generar o recolectar los datos. Los flujos del *dataset* están definidos en formato XML (*eXtensible Markup Language*), por lo que tendremos que realizar un proceso de lectura y transformación para poder importar los datos correctamente. Para evitar la repetición de este proceso, podemos almacenar el conjunto de datos resultante en un archivo con formato CSV (*Comma-Separated Values*), para cuando sea preciso simplemente importar este archivo.

4.3 Exploración

Como se comentó en la sección 4.1, el *dataset* tiene una gran cantidad de datos, lo cual podía llegar a ser una carga computacional muy alta, incluso pudiendo algunas tareas ser imposibles de llevar a cabo con un ordenador personal, por lo que en un principio se trabajó solo con los datos del día trece. Posteriormente según se fue avanzando en las fases del proyecto

se observó que computacionalmente era viable trabajar con todo el *dataset*, por lo que en este apartado comentaremos los análisis hechos con los datos de todos los días.

En un primer análisis, como podemos observar en la figura 4.2, observamos que contiene 1.938.464 flujos y cada uno tiene 20 características. Lo siguiente es ver cómo se distribuye

```
Int64Index: 1938464 entries, 0 to 61831
Data columns (total 20 columns):
 #   Column                          Dtype
---  ------                          -----
 0   appName                         object
 1   totalSourceBytes                object
 2   totalDestinationBytes           object
 3   totalDestinationPackets         object
 4   totalSourcePackets              object
 5   sourcePayloadAsBase64           object
 6   sourcePayloadAsUTF              object
 7   destinationPayloadAsBase64      object
 8   destinationPayloadAsUTF         object
 9   direction                       object
 10  sourceTCPFlagsDescription       object
 11  destinationTCPFlagsDescription  object
 12  source                          object
 13  protocolName                    object
 14  sourcePort                      object
 15  destination                     object
 16  destinationPort                 object
 17  startDateTime                   object
 18  stopDateTime                    object
 19  Tag                             object
dtypes: object(20)
memory usage: 3.3 GB
```

Figura 4.2: Información de las características iniciales del *dataset*.

la clase objetivo, y tras observar la figura 4.3 vemos que se produce un gran desbalanceo ya esperado, puesto que lo normal es que haya más tráfico normal que de ataque. En este caso, el 96.55% son flujos benignos y el resto son flujos de ataque. Estos dos aspectos pueden suponer dos problemas que afrontaremos más adelante en la sección (5.1): una gran carga computacional al tener un conjunto de datos tan grande, y el alto desbalanceo puede ocasionar problemas de sobreajuste al intentar crear los modelos.

Otra de las tareas en esta fase es ver si el *dataset* está completo, o lo que es lo mismo, ver el número de valores no disponibles. En un análisis profundo apreciamos lo siguiente:

- *destinationPayloadAsUTF* - 58.0 % valores perdidos
- *destinationPayloadAsBase64* - 58.0 % valores perdidos

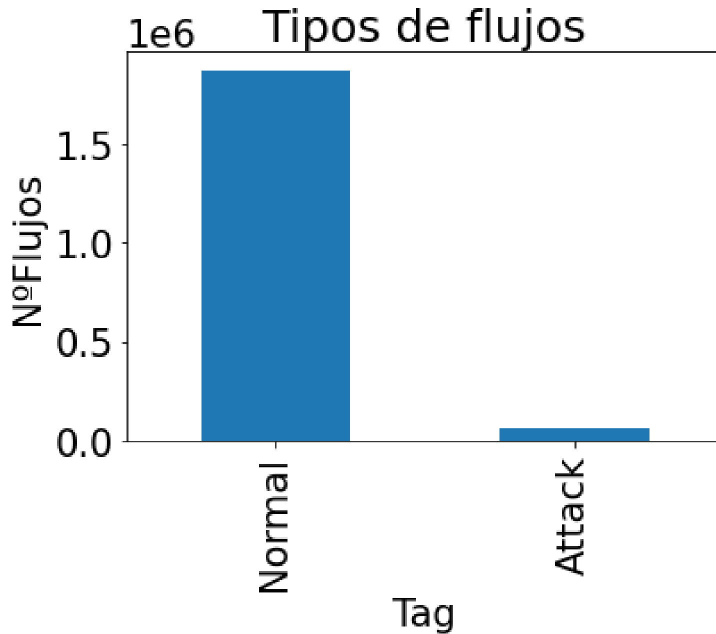


Figura 4.3: Cantidad de flujos normales y de ataques.

- *sourcePayloadAsUTF* - 57.8 % valores perdidos
- *sourcePayloadAsBase64* - 53.4 % valores perdidos
- *destinationTCPFlagsDescription* - 3.2 % valores perdidos
- *sourceTCPFlagsDescription* - 0.2 % valores perdidos

Los flujos con valores no disponibles en los atributos relacionados al *payload* serían la mitad del *dataset*, un número elevado pero normal debido a que los paquetes no tienen por qué tener *payload*. Como es un número muy alto de valores perdidos y tampoco aporta información relevante hemos decidido prescindir de estas características.

En cuanto a los otros atributos, los relacionados con los *flags* de TCP (*Transmission Control Protocol*), en la figura 4.4 podemos ver cómo se distribuyen, observando que hay un alto número de flujos cuyo valor es ”N/A”. Analizando más en profundidad vemos que el significado de este valor es para indicar que el flujo TCP no lleva ningún *flag* de este tipo activo (puesto que el protocolo TCP no obliga a tener que llevar siempre un *flag* TCP activado), o que también se puede tratar de un flujo perteneciente a otro protocolo, en cuyo caso es normal que no haya ningún *flag* de este tipo activo. Por lo tanto, este valor ”N/A” nos servirá más adelante, a la hora de realizar la transformación de los datos (sección 5.2), para contemplar estos dos casos comentados.

```
F,S,P,A                       1102213
N/A                            389525
S,P,A                          172511
F,A                            122657
S                               41484
S,R,P,A                         25726
F,P,A                           24721
P,A                             16592
A                               11704
F,S,A                            9489
F,S,R,P,A                        7954
F,R,A                            4551
S,A                              3383
R,A                              1052
R                                 838
S,R                               544
F,R,P,A                           128
R,P,A                             107
S,R,A                              25
S,R,Illegal7,Illegal8              18
F,P,U                              12
F,S,R,A                            12
F,S,R,P,U                          10
F,S,P,U                             2
Name: sourceTCPFlagsDescription, dtype: int64
```

Figura 4.4: Información la característica *sourceTCPFlagsDescription*.

El resto de los *flags* vemos que aparecen en la especificación del protocolo TCP, salvo el Illegal7 e Illegal8, por lo que en la siguiente etapa los flujos que contengan estos *flags* se eliminarán, al ser estos valores no válidos.

Capítulo 5

Preparación de los datos

En esta fase el objetivo principal es obtener un buen conjunto de datos capaz de crear modelos cuyos resultados sean lo mejor posible. Para esto, sobre el conjunto de datos inicial se realizan una serie de tareas, las cuales comentaremos en las siguientes secciones.

5.1 Limpieza de los datos

En esta fase de preparación de los datos, una tarea muy común es la de la limpieza de los valores atípicos aplicando técnicas cómo pueden ser el diagrama de cajas, DBSCAN (DBSCAN) o *Isolation Forest* para encontrar y eliminar los valores anómalos, puesto que en función del algoritmo que se utilice puede que los *outliers* perjudiquen en el resultado.

El diagrama de cajas es un método muy usado y sencillo de entender. Se basa en el rango intercuartílico y en la representación gráfica de los diagramas de cajas. Estas gráficas muestran el resumen de cinco estadísticas sobre el conjunto de datos, que son: el mínimo, el primer cuartil, la mediana, el tercer cuartil y el máximo. En la figura 5.1 podemos ver la forma de los diagramas de cajas. En cuanto al rango intercuartílico (IQR), este se refiere a la distancia entre

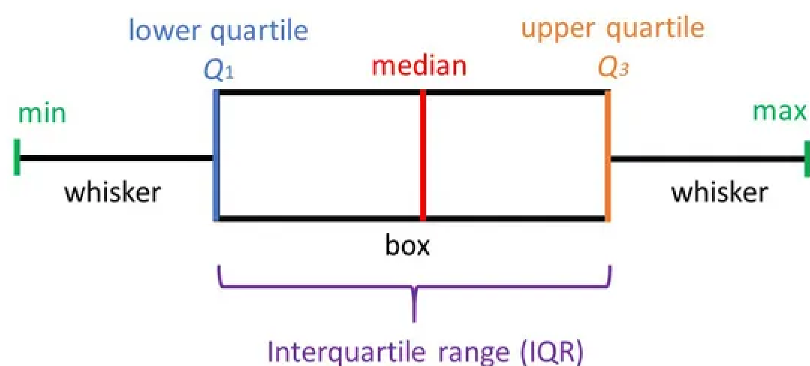


Figura 5.1: Ejemplo de diagrama de cajas.

el primer y tercer cuartil, definiendo la posición de los ”bigotes de la caja”. Los datos que se encuentren fuera de los bigotes son los que se consideran valores atípicos [36].

El segundo método mencionado es DBSCAN (*Density-based spatial clustering of applications with noise*), un algoritmo de *clustering* que también se puede utilizar para la detección de datos anómalos basado en la densidad con datos unidimensionales o multidimensionales. Tiene tres aspectos importantes:

- Puntos centrales (*Core Points*): punto que forma un grupo al tener al menos *min_samples* dentro de una distancia *epsilon*.

- Puntos fronterizos (*Border Points*): punto que está en el mismo grupo que un punto central pero lejos del centro.

- Punto anómalo (*Noise Points*): punto que no pertenece a ningún grupo, esto es, que no contiene al menos *min_samples* puntos que estén a una distancia *epsilon* de él ni se encuentra a una distancia *epsilon* de un punto central, por lo que puede ser un valor atípico.

Este método requiere que el conjunto de datos esté estandarizado o escalado, y los valores de *min_samples* y *epsilon* deben ser escogidos por nosotros [37][38].

El último método, *Isolation Forest*, es un algoritmo perteneciente a la familia de los árboles de decisión. La idea principal, y que lo diferencia de otros métodos de detección de *outliers*, es que este algoritmo trata de identificar explícitamente las anomalías en lugar de caracterizar los datos normales. Esta diferencia la podemos ver en la figura 5.2, demostrando que para identificar una muestra anómala hacen falta menos particiones que para identificar una muestra normal [39].

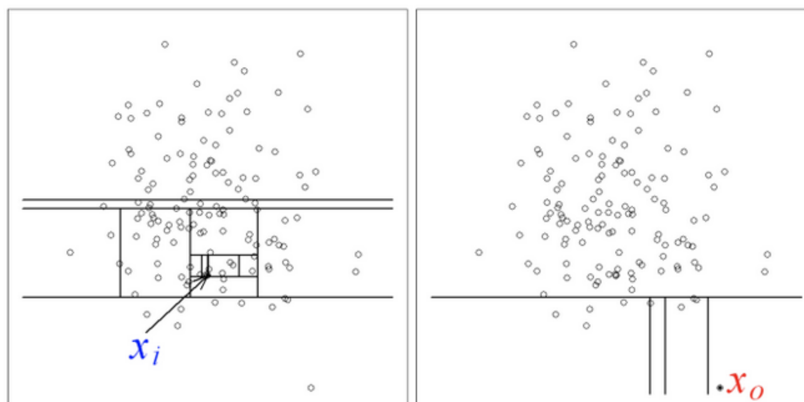


Figura 5.2: Diferencia entre identificar una muestra normal (izquierda) y una anómala (derecha).

Estos métodos se han visto de forma teórica y se ha decidido no utilizarlos, puesto que el objetivo que tenemos encomendado es ese, encontrar datos atípicos que nos ayuden a identificar los flujos de ataque, por lo que aplicar un estudio exhaustivo para la limpieza de los datos podría ser contraproducente al poder eliminar flujos clasificados como de Ataque.

Es por este motivo, por el que nos declinamos por no eliminar valores detectados como atípicos por estos algoritmos, al poder ser valores correctamente medidos, pero lo que sí tendremos en cuenta, guiándonos por el artículo sobre el *dataset*[40], son los flujos cuya dirección es R2R (*Remote-to-Remote*) y las direcciones IP 0.0.0.0, puesto que no se nombran en el artículo y entendemos que esos flujos no se deberían de haber capturado o son erróneos, por lo que se eliminarán.

En esta tarea también incluimos el balanceo de las dos clases objetivo. Como vimos en la anterior sección, hay un gran desbalanceo entre la clase Normal y la clase Ataque. Para solucionar este desbalanceo se pueden aplicar tres técnicas [41]:

- *Oversampling*: consiste en añadir más muestras a la clase minoritaria, ya sea repitiendo muestras aleatoriamente o generando otras nuevas a partir de las ya existentes.

- *Undersampling*: consiste en reducir las muestras de la clase mayoritaria, pudiendo realizarse de forma aleatoria.

- *Stratification*(Estratificación): consiste en muestrear un conjunto de datos en subconjuntos, manteniendo la misma proporción de cada variable.

Descartamos el *oversampling* debido a la naturaleza de los datos, puesto que sería muy difícil crear nuevas muestras a partir de las ya existentes, por lo que consideramos como mejor opción aplicar *random undersampling*.

Para esto ya existen librerías como Imbalanced-Learn que tienen algoritmos para hacer el balanceo, pero en este caso queremos hacer un balanceo un poco más riguroso que solo centrándonos en la clase mayoritaria. Para esto, vamos a tener en cuenta el protocolo, la dirección y si el flujo es normal o de ataque, por lo que creamos una característica que indique el tipo de flujo teniendo en cuenta estos requisitos. A partir de aquí aplicamos *random undersampling* manualmente, contando el número de muestras que son flujos de tipo normal y los que son de tipo ataque teniendo en cuenta la nueva característica creada, y entre las muestras de esas dos clases, eliminamos las que pertenezcan a la clase mayoritaria. Tras aplicar esta técnica con todos los tipos de flujos ya tendríamos las clases Normal y Ataque balanceadas de una forma más estricta.

El *undersampling* aplicado mantiene una proporción 1:1, por lo que en este caso el número total de muestras se reduce drásticamente, puesto que de 1.938.464 muestras pasamos a 127.512, debido a que en todo el conjunto de datos sólo hay 63.756 flujos de ataque. Este

proceso lo podríamos aplicar ahora o como paso previo a la selección de las características. Se decide hacer en este punto, porque tal y como veremos más adelante, la creación de algunas características tiene una alta carga computacional, y con esta gran reducción de la cantidad de datos estaremos ayudando a reducir la carga computacional y el tiempo de ejecución.

5.2 Transformación

En esta fase realizaremos los procesos necesarios para extraer la mayor cantidad de información posible a partir de los datos que tenemos en el *dataset* creando nuevas características en algunos casos y, en otros, adecuaremos las características ya existentes para usarlas en los entrenamientos de los algoritmos.

También realizaremos en esta fase el subtipado, un proceso bastante importante en términos de rendimiento.

5.2.1 Creación de características

Esta tarea posiblemente sea la que más tiempo requiere en estas primeras fases, por la necesidad de analizar bien los datos que tenemos y pensar en cómo a partir de ellos extraer más conocimiento, aplicando lo que sabemos sobre el escenario en el que se generaron y también el conocimiento que tenemos sobre el propio ámbito de estudio.

Lo primero es comprobar si los atributos que ya tenemos los podemos utilizar tal cual están y si nos proporcionan información. Para esto, unas de las características que más llaman la atención son las referentes a las fechas, donde su comprensión resulta algo difícil pero aportan una información muy concreta la cual podría dar lugar al *overfitting*, por lo que creamos la característica *duración*, que muestra información más comprensible y que debería ayudar a los algoritmos a generalizar mejor.

El siguiente paso puede ser tratar las características que son cadenas de caracteres, puesto que los algoritmos como Random Forest o Regresión Logística solo trabajan con características numéricas. De esas características hay algunas como *direction* o *protocolName* que vemos que son variables categóricas, y para convertirlas a numéricas podemos usar lo que se denomina *One-Hot-Encoding*, para crear tantas variables binarias como clases haya en esa variable, teniendo en cuenta la multicolinealidad a la hora de realizar este paso con el método *get_dummies* de la librería Scikit-Learn.

Respecto al resto de variables que son texto, como las variables relacionadas con los *TCPFlags* o con las direcciones IP, tenemos que aplicar un proceso manual aplicando expresiones regulares para obtener las correspondientes características binarias. En cuanto a las variables de los *TCPFlags*, aplicamos las expresiones regulares para saber qué *flags* van marcados en el texto para establecer a uno las correspondientes características. En lo referente a las varia-

bles de las direcciones IP, aplicamos también expresiones regulares para ver con qué patrón se corresponde la IP. Así sabremos a qué LAN del escenario pertenece y marcaremos a uno la respectiva variable binaria de las que hemos creado, teniendo una característica por cada LAN, diferenciando origen y destino.

Respecto a las características iniciales, ya se podrían usar todas directamente y aportando suficiente información, aunque aún se puede hacer una modificación más, que consiste en convertir las variables *sourcePort* y *destinationPort* en variables categóricas, utilizando el rango de puertos marcado por la IANA:

- Conocidos : 0 - 1023
- Registrados: 1024 - 49151
- Dinámicos: 49152 - 65535

Este proceso se llevó a cabo, pero según fueron avanzando los entrenamientos de los algoritmos se observó que determinados ataques estaban muy ligados a números de puertos concretos, por lo que si añadiésemos estas variables categóricas y suprimiésemos las variables iniciales estaríamos eliminando información, por lo que al final se decidió por prescindir de este paso.

En la siguiente iteración se añadieron nuevas características derivadas de algunas de las más importantes como son las relacionadas con el número de *bytes* y paquetes, dando lugar a las tasas, puesto que en el tráfico de red se suelen usar más las tasas que los acumulados:

- *BytesPerSecond*: número de bytes entre la duración del flujo.
- *PacketsPerSecond*: número de paquetes entre la duración del flujo.
- *BytesPerPackets*: número de bytes entre el número de paquetes.
- *IOPR*: número de paquetes de origen entre el número de paquetes de destino.
- *IOBR*: número de bytes de origen entre el número de bytes de destino.

Algo a tener en cuenta es que muchos flujos tienen como duración cero segundos. Esto se debe a que la duración no alcanza los 0,50 segundos y al redondear el resultado es cero. Es por este motivo por el que en las variables resultado de una división en la que el divisor es la duración, si el valor de éste es cero se sustituyó por 0,25, por ser el valor medio entre 0 y 0,50, para de esta forma intentar no desvirtuar los datos.

Otras características adicionales que se añaden son las categóricas que nos indican en qué momento del día se inicia el flujo:

- Inicio por la mañana [8h - 13h]

- Inicio al mediodía [13h - 16h]
- Inicio por la tarde [16h - 21h]
- Inicio por la noche [21h - 8h]

Ya para terminar con la creación de nuevos atributos, procedemos a crear aquellos que pueden requerir un alto coste computacional debido al gran número de operaciones que se deben realizar. Para entender este proceso tenemos que recordar la definición de tipo de flujo que hicimos anteriormente en esta sección, donde considerábamos que los flujos eran del mismo tipo si tenían el mismo protocolo, dirección y etiqueta (normal o ataque). Teniendo en cuenta esto, el proceso consistirá en seleccionar cada uno de los flujos, calcular la diferencia de sus valores numéricos con todos y cada uno del resto de los flujos, por un lado con los flujos que son normales y por otro lado con los flujos que son de ataque, y por último calcular el máximo, mínimo, media, mediana, varianza y desviación típica de cada una de las diferencias calculadas para cada atributo. De esta forma tendríamos la media de las diferencias de *totalSourcePakets* para los flujos normales y por otro lado para los de ataque, y así sucesivamente con el resto de medidas y de características.

De esta forma, para cada característica numérica estaríamos creando otras doce nuevas características, siendo un total de ciento treinta y dos nuevas características.

Como ya se comentó anteriormente, este es un proceso muy costoso que requiere ejecutarse tantas veces como muestras tenga el *dataset*, siendo el motivo por el que se hizo *undersampling* ya en una etapa bastante temprana.

Cabe destacar también que, para reducir el tiempo de ejecución de este proceso, hemos aplicado técnicas de paralelización.

5.2.2 Escalado de los datos

En un conjunto de datos como este, con una gran cantidad de muestras y variables, lo más normal es que los datos estén en diferentes magnitudes, unidades y rangos. Esto a veces puede reducir la capacidad de predicción en algunos algoritmos que se basan en la distancia Euclídea entre dos puntos de datos, debido a que en los cálculos las características con altas magnitudes pesarán mucho más que las características con bajas magnitudes, por lo que tomarán un papel más determinante a la hora del entrenamiento [42].

Otros dos motivos por los que puede ser necesario aplicar el escalado de los datos es que algunos algoritmos, como el descenso del gradiente de redes neuronales, convergen más rápido con los datos escalados que sin escalar. También en redes neuronales, en el caso de la activación sigmoidea, el escalado ayudaría a no saturarse demasiado rápido [43].

En nuestro caso, la razón principal por la que nos interesa escalar los datos es por el uso del algoritmo de Regresión Logística, puesto que a *Random Forest* no le afecta tanto que los datos

estén sin escalar. En cuanto a *Naive Bayes*, más adelante veremos los motivos pero también nos interesa escalar los datos, en este caso estandarizándolos obteníamos mejores resultados, por lo que haremos un procesamiento diferente para este algoritmo.

Para realizar la tarea de escalado podemos elegir entre distintos algoritmos, donde para algunos hay que tener en cuenta la distribución de los datos, si hay algún dato anómalo o no, etc. En este caso nos hemos decantando por probar *MinMaxScaler* y *RobustScaler* para comprobar cuál arrojaba mejores resultados, siendo el ganador *MinMaxScaler*, por lo que utilizaremos este mecanismo para escalar los datos.

En cuanto a la estandarización de los datos para usarlos con *Naive Bayes*, haremos uso de *StandardScaler*, donde como resultado tendremos que la media de los valores observados será cero y la desviación estándar uno.

5.3 Selección de las características

Como último paso en esta fase del proyecto, nos encontramos con una tarea muy importante de cara a los resultados finales que consiste en seleccionar las mejores características en función del algoritmo que utilicemos.

Tras realizar la tarea de creación, nos encontramos con un total de ciento ochenta características, lo que daría un conjunto de datos de 127.512 filas por ciento ochenta columnas, un conjunto bastante amplio de datos. Este número de datos puede ser demasiado extenso y perjudicial, ya no solo en términos de rendimiento, sino también en la precisión de los modelos creados al usar características irrelevantes que influyen negativamente [44]. Es por esto por lo que se realiza la selección de características, un proceso manual o automático en el que se escogen las características que más influyen en la variable a predecir, teniendo como ventajas la reducción del sobreentrenamiento, mejora de la precisión y reducción del tiempo de entrenamiento.

En cuanto a los métodos automáticos, los podemos dividir en tres grupos [45]:

- Filtrado: usan técnicas estadísticas para evaluar la relación entre cada variable de entrada y la variable objetivo, y esta evaluación se utiliza como base para filtrar las características que se usarán en el modelo.

- *Wrapper*: estos métodos crean un amplio número de modelos con diferentes subconjuntos de características, y seleccionan las que dan como resultado el mejor modelo teniendo en cuenta alguna métrica. Estos métodos pueden tener una cierta carga computacional.

- *Intrinsic* o *embedded methods*: son algoritmos que como parte del proceso de aprendizaje ya realizan la selección de características. Combinan las cualidades de los métodos de

filtrado y *wrapper*. En este grupo se incluyen algoritmos como *Lasso* o *Random Forest*, entre otros.

Antes de aplicar alguno de estos métodos, cabe destacar que como primer paso en la selección de características siempre debemos aplicar nuestro conocimiento adquirido sobre el *dataset*, como puede ser el significado de las características o las correlaciones entre ellas. Es por esto por lo que primeramente podemos echar un vistazo a la matriz de correlaciones de algunas de las principales características (figura 5.3), donde podemos ver que algunas variables están bastante correlacionadas, llamando especial atención la correlación entre *totalSourceBytes-totalDestinationBytes* y *totalSourcePackets-totalDestinationPackets* pudiendo ser este comportamiento por usar un *dataset* sintético, puesto que en un escenario real no se tiene por qué dar esta situación. Ante una correlación tan alta, el proceso a seguir sería suprimir una de

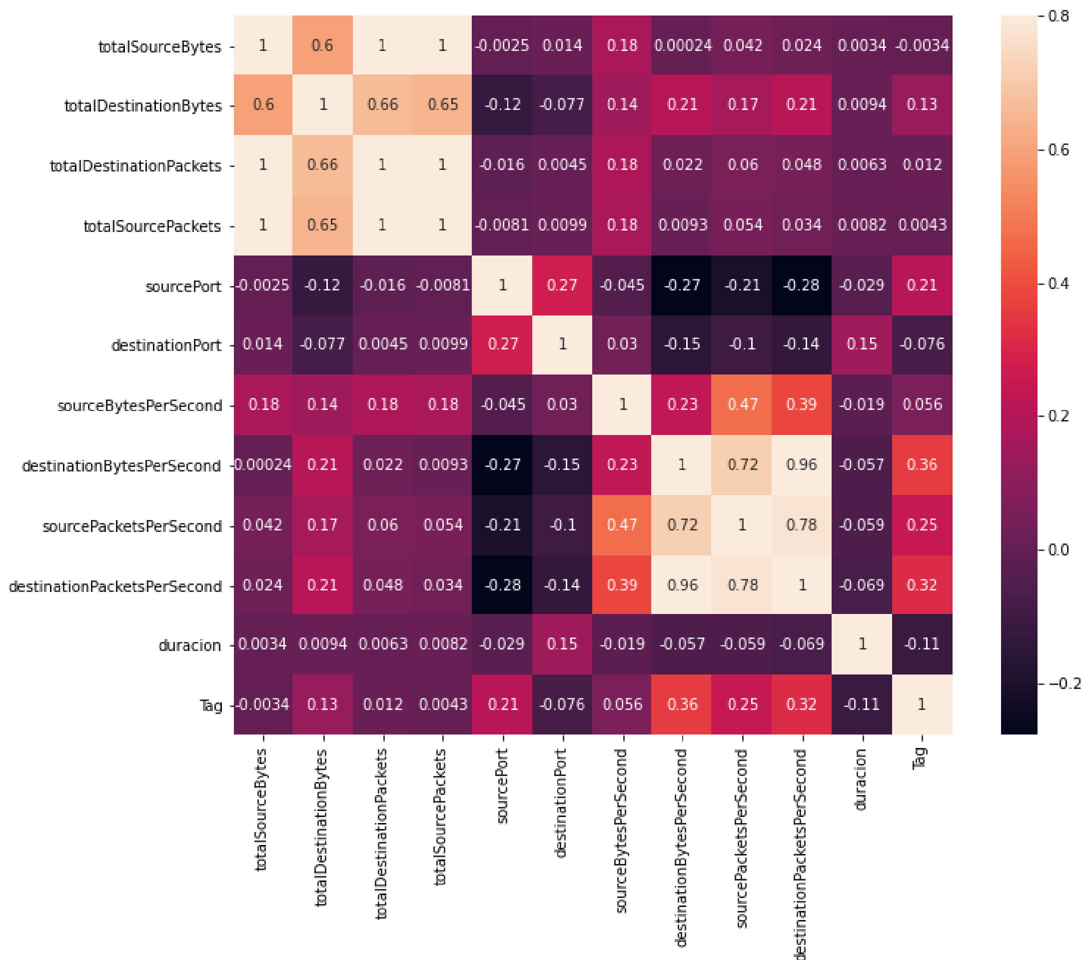


Figura 5.3: Matriz de correlaciones con algunas de las principales característica.

las características, pero viendo que estas son bastante importantes, primero hacemos una pe-

queña prueba para ver los resultados usando estas características y otra eliminándolas, y los resultados son mejores sin eliminarlas.

Salvo esta excepción, con las demás características procedemos a eliminarlas. Ante el alto número de variables de las que disponemos, sería inviable visualizarlas todas en la matriz de correlaciones para poder eliminarlas. Es por esto por lo que creamos una función que pasándole el *dataset* y un umbral, nos devuelva una lista con las características cuya correlación superen el umbral, facilitándonos así el trabajo de eliminación de altas correlaciones.

5.3.1 Regresión Logística

El algoritmo de Regresión Logística es unos de los más sensibles ante la presencia de datos correlacionados, pero hecho el paso anteriormente comentado ya podemos pasar a usar algún algoritmo para la selección de características, donde de entre las muchas posibilidades que ofrece Scikit-Learn nos quedamos con RFECV (*Recursive Feature Elimination with Cross-Validation*), el cual es un método perteneciente al grupo de los *wrapper based.* Este método usa RFE (Eliminación Recursiva de Características) para seleccionar recursivamente las mejores características teniendo en cuenta el atributo **coef_** o **feature_importances_** y eliminando las menos importantes hasta obtener el número óptimo de mejores características, el cual se obtiene ejecutando la validación cruzada (CV).

Tras ejecutar el algoritmo RFECV, obtenemos que el número óptimo de características a usar es cincuenta y uno. Al analizarlas podemos observar que la mayoría tuvieron un alto coste de cálculo, a partir de las diferencias, pero vemos que ese esfuerzo adicional fue fructífero. En la tabla 5.1 podemos ver las características seleccionadas y su importancia.

Tabla 5.1: Resumen de las características seleccionadas para Regresión Logística y su importancia.

CARACTERÍSTICAS REGRESIÓN LOGÍSTICA	
CARACTERÍSTICA	IMPORTANCIA
SourceFlagS	11.637694
DiffStandardDeviationDestinationBytesPerPacket...	10.221160
SourceFlagA	9.309078
destinationBytesPerSecond	6.381755
DiffMinDestinationBytesPerPacketNormal	5.191874
inicioMediodia	5.026894
SourceFlagU	4.482652
inicioTarde	4.319968
Continuar en la siguiente página	

Tabla 5.1 – continuación de la página anterior

CARACTERÍSTICA	IMPORTANCIA
DiffMinSourceBytesPerPacketNormal	3.893653
SourceFlagR	3.314245
DestinationFlagA	3.122151
DiffMaxSourceBytesPerPacketAttack	2.982961
destinationLAN3	2.767015
destinationBytesPerPacket	2.590156
sourceLAN4	2.469575
sourceLAN1	2.454138
destinationLAN2	2.410167
direction_R2L	2.235342
protocol_udp_ip	1.708095
destinationPort	1.388269
direction_L2L	1.382661
destinationLAN1	1.202039
IOPR	1.128683
protocol_tcp_ip	1.123236
SourceFlagF	1.053097
totalDestinationBytes	0.992295
sourcePort	0.708822
destinationLAN4	0.563412
DiffMinTotalSourceBytesNormal	0.404158
DiffMinDuracionNormal	0.272106
DiffMaxTotalSourcePacketsAttack	-0.465719
DiffMinIOPRAttack	-0.681287
DiffMinDestinationBytesPerSecondAttack	-0.708183
DiffMinDestinationBytesPerPacketAttack	-1.243183
SourceFlagP	-1.336316
destinationLAN5	-1.338868
DiffMaxDuracionAttack	-1.746012
sourceBytesPerPacket	-1.915181
sourceBytesPerSecond	-2.182062
sourceLAN5	-2.936480
protocol_icmp_ip	-3.049193
direction_L2R	-3.835865

Continuar en la siguiente página

Tabla 5.1 – continuación de la página anterior

CARACTERÍSTICA	IMPORTANCIA
DiffStandardDeviationSourceBytesPerSecondAttack	-3.941618
DestinationFlagR	-6.060430
inicioMañana	-6.260805
DiffMinSourceBytesPerPacketAttack	-7.166651
DiffVarianceSourceBytesPerPacketNormal	-8.831012
DestinationFlagP	-9.113597
DiffMedianDestinationBytesPerPacketAttack	-10.482541
DiffMeanDestinationBytesPerPacketAttack	-11.529668
DestinationFlagS	-13.865484

5.3.2 *Random Forest*

Con *Random Forest*, aunque ya es un algoritmo bastante preciso y que se encuentra dentro del grupo de algoritmos que internamente ya realiza selección de características, ejecutaremos también RFECV para ver el número óptimo de características y cuáles selecciona.

Tras su ejecución, algo lenta, se puede observar que arroja unos resultados muy buenos, siendo treinta el número óptimo de características a utilizar. Tras probar el algoritmo con las características seleccionadas, vemos que los resultados son levemente mejores que los obtenidos sin realizar selección de características, aunque si tenemos en cuenta el poco margen de mejora del que se disponía, podemos concluir este resultado como satisfactorio.

Las características seleccionadas y su importancia las podemos ver en la tabla 5.2.

5.3.3 *Naive Bayes*

Para *Naive Bayes* este paso es un poco más complicado que en los casos anteriores al requerir una serie de pasos extras. Esto se debe a los requisitos de las diversas implementaciones que dispone *Naive Bayes* y también a los distintos tipos de datos que contiene el *dataset*, habiendo variables numéricas y categóricas, por lo que no podemos usar directamente todos los datos como entrada de alguno de los tres algoritmos de *Naive Bayes* que ofrece Scikit-Learn.

Esto se debe a que, como bien se explicó anteriormente, en función de los tipos de datos debemos aplicar una u otra implementación. La librería implementa Guassian Naive Bayes, que se suele usar para datos continuos, y Bernoulli Naive Bayes, que se suele utilizar con datos discretos, además de Multinomial Naive Bayes. Ante esta situación debemos aplicar una metodología más sofisticada que la aplicada con Regresión Logística o *Random Forest*, en la

CARACTERÍSTICAS RANDOM FOREST	
CARACTERÍSTICA	IMPORTANCIA
inicioMediodia	0.122371
DiffMeanDestinationBytesPerPacketAttack	0.089275
IOPR	0.059641
DiffMinDestinationBytesPerPacketAttack	0.058254
totalDestinationBytes	0.057344
SourceFlagA	0.054858
totalDestinationPackets	0.049988
totalSourcePackets	0.047958
DiffMaxTotalSourcePacketsAttack	0.044781
destinationBytesPerSecond	0.041835
DiffMinDestinationBytesPerSecondAttack	0.034700
DiffMaxSourceBytesPerPacketAttackDiffStandardDeviationDestination	0.031278
DiffMedianDestinationBytesPerPacketAttack	0.030443
sourcePort	0.026046
DiffVarianceSourceBytesPerPacketNormal	0.025804
DestinationFlagP	0.023217
destinationBytesPerPacket	0.018820
DiffMaxSourceBytesPerPacketAttack	0.018356
destinationPort	0.017958
DiffStandardDeviationSourceBytesPerSecondAttack	0.017607
sourceLAN2	0.017467
DiffMinSourceBytesPerPacketAttack	0.016019
sourcePacketsPerSecond	0.015427
sourceBytesPerPacket	0.015305
sourceBytesPerSecond	0.014898
inicioNoche	0.012073
DiffMinSourceBytesPerPacketNormal	0.011996
totalSourceBytes	0.010688
DiffMinTotalDestinationBytesAttack	0.007831
SourceFlagF	0.007763

Tabla 5.2: Resumen de las características seleccionadas para *Random Forest* y su importancia.

que tenemos dos alternativas: convertir todos los datos al mismo tipo, ya sea a datos numéricos o binarios y utilizar un único algoritmo, o tratar los distintos tipos de datos de forma separada aplicando la implementación del algoritmo apropiada y juntando posteriormente los resultados obtenidos para aplicar uno de los dos algoritmos cuya naturaleza son los datos continuos.

La opción escogida fue la segunda, donde el proceso, entrando en más detalle, es el siguiente:

- De forma independiente se entrena la parte continua del *dataset* aplicando GaussianNB y la parte categórica aplicando BernoulliNB.
- Se calcula la probabilidad de asignación a cada clase aplicando el método *predict_proba* para ambas partes (continua y discreta).
- Con las probabilidades creadas anteriormente se crean nuevas características transformando el *dataset* completo.
- De esta forma ya tenemos todo el *dataset* con datos continuos, por lo que se puede volver a entrenar el modelo aplicando GaussianNB.

Por lo tanto para *Naive Bayes* hay que crear dos modelos intermedios para poder crear correctamente el modelo final. En esos modelos intermedios aplicamos selección de características, donde a diferencia de Regresión Logística o *Random Forest* no podemos hacer uso de los atributos **coef_** o **feature_importances_** necesarios para aplicar RFECV. Es por este motivo por lo debemos buscar otro método, eligiendo en un primer momento PCA (*Principal Component Analysis*), un método muy usado para la reducción de dimensionalidad, puesto que nos permite calcular de una forma cómoda el mejor número de componentes a utilizar en el entrenamiento del modelo.

El problema de utilizar este método es que no nos permite ver la importancia de las características seleccionadas, por lo que, para realizar todos los pasos seguidos con los otros dos modelos, debemos usar otra alternativa. Es por esto por lo que decidimos usar *Random Forest* para la selección de características, siendo este un método *Embedded*.

En este caso, para extraer automáticamente las características seleccionadas se utiliza el método *selectFromModel*, el cual se encarga de extraer las mejores características basándose en el peso de las importancias. A este método se le puede especificar el parámetro *threshold* (umbral), el cual se utiliza para seleccionar las características cuya importancia superen el umbral. Si no se especifica nada, por defecto utiliza la media.

Ejecutando este método para las características numéricas, se escogen dieciséis de las treinta y tres que dispone el conjunto de datos, pudiendo ver cuáles son y su importancia en la tabla 5.3.

CARACTERÍSTICAS GAUSSIAN NAIVE BAYES	
CARACTERÍSTICA	IMPORTANCIA
totalSourceBytes	0.046104
totalDestinationBytes	0.041467
totalDestinationPackets	0.042781
totalSourcePackets	0.041787
sourcePort	0.032707
destinationBytesPerSecond	0.041457
destinationBytesPerPacket	0.042162
IOPR	0.062765
DiffMaxTotalSourcePacketsAttack	0.036051
DiffMinTotalDestinationBytesAttack	0.056505
DiffMinDestinationBytesPerSecondAttack	0.080164
DiffMedianDestinationBytesPerPacketAttack	0.041614
DiffMinDestinationBytesPerPacketAttack	0.052961
DiffMeanDestinationBytesPerPacketAttack	0.101603
DiffStandardDeviationDestinationBytesPerPacketAttack	0.036736
DiffVarianceSourceBytesPerPacketNormal	0.043006

Tabla 5.3: Resumen de las características seleccionadas para Gaussian Naive Bayes y su importancia.

En cuanto a las características categóricas, se escogen once de las treinta y uno, pudiendo verlas en la tabla 5.4

CARACTERÍSTICAS BERNOULLI NAIVE BAYES	
CARACTERÍSTICA	IMPORTANCIA
SourceFlagS	0.037119
SourceFlagF	0.062318
SourceFlagA	0.037643
SourceFlagP	0.037341
DestinationFlagR	0.039020
DestinationFlagP	0.046866
destinationLAN5	0.033005
inicioMañana	0.090393
inicioMediodia	0.350492
inicioTarde	0.038580
inicioNoche	0.085158

Tabla 5.4: Resumen de las características seleccionadas para Bernoulli Naive Bayes y su importancia.

Capítulo 6

Modelado

En esta fase se seleccionan las técnicas de modelado más apropiadas para el trabajo que se está realizando, ajustando los parámetros necesarios para obtener unos resultados óptimos, y en caso de ser necesario se regresa a la fase anterior para ajustar los datos correctamente. Finalmente se entrenan y validan los modelos creados.

El objetivo principal de esta fase es obtener el mejor modelo posible. Para esto se llevan a cabo una serie de pasos, de los cuales algunos como el escalado o la selección de características ya han sido abordados anteriormente, quedando por realizar el ajuste de los parámetros, el entrenamiento y validación de los modelos obtenidos.

6.1 Validación

En este apartado definiremos la forma de validar el modelo y las métricas empleadas.

6.1.1 Métricas de rendimiento

Una de las mejores formas de interpretar el rendimiento de un algoritmo de clasificación es mediante la matriz de confusión [46], que tiene una forma como la de la figura 6.1, donde sus elementos son los siguientes:

- VP (Verdaderos Positivos): cantidad de elementos positivos clasificados correctamente como positivos.
- VN (Verdaderos Negativos): cantidad de elementos negativos clasificados correctamente como negativos.
- FN (Falsos Negativos): cantidad de elementos positivos clasificados incorrectamente como negativos.

		Predicción	
		Positivos	Negativos
Observación	Positivos	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

Figura 6.1: Matriz de confusión.

- FP (Falsos Positivos): cantidad de elementos negativos clasificados incorrectamente como positivos.

A partir de estos valores podemos obtener unas métricas más concisas [47], que son las siguientes:

- Precisión: se refiere a la dispersión del conjunto de valores obtenidos a partir de mediciones repetidas de una magnitud. Cuanto menor es la dispersión mayor será la precisión. La forma de calcularla es:

$$PRECISION = \frac{VP + VN}{VP + VN + FP + FN}$$

- Sensibilidad o *Recall*: ligada a la precisión también se suele usar la métrica llamada *recall* o Tasa de Verdaderos Positivos (TPR). La forma de calcularla es:

$$RECALL = \frac{VP}{VP + FN}$$

- Exactitud: se refiere a lo cerca que está el resultado de una medición del valor verdadero. Está relacionada con el sesgo de una estimación. La forma de calcularla es:

$$EXACTITUD = \frac{VP}{VP + FP}$$

- Especificidad: se refiere a los casos negativos que se clasificaron correctamente. También denominada Tasa de Verdaderos Negativos. La forma de calcularla es:

$$ESPECIFICIDAD = \frac{VN}{VN + FP}$$

- F1-score: combina la precisión y la sensibilidad en una sola métrica, pudiendo usarse como una forma simple de comparar dos clasificadores. La forma de calcularla es [48]:

$$F1 = \frac{VP}{VP + \frac{FN+FP}{2}}$$

Otra métrica importante a utilizar puede ser el índice Kappa (*Cohen's Kappa Coefficient*), que lo que hace, en una breve descripción, es medir el grado de acuerdo entre los estimadores. Es una métrica bastante sofisticada puesto que no solo tiene en cuenta el grado de acuerdo entre los estimadores, sino que también tiene en cuenta la proporción esperada de acuerdos por casualidad. De esta forma, su valor puede estar entre cero y uno, donde la puntuación cercana al cero significa que hay un acuerdo aleatorio entre los evaluadores, y una puntuación cercana al uno significa que hay acuerdo total. También se puede dar una puntuación negativa, significando que hay menos acuerdo que oportunidad [49][50].

Mostraremos el resultado de todas las métricas, aunque nos fijaremos más en la precisión a la hora de tomar decisiones. Escoger una métrica en lugar de otra viene determinado por la naturaleza del trabajo y por el tamaño de error de clasificación que estemos dispuestos a permitir. En este caso es peor un falso negativo frente a un falso positivo (clasificar un ataque como flujo normal, frente a clasificar un flujo normal como ataque). Es por esto último, y ante la ventaja de que ya tenemos el *dataset* balanceado por lo que escogemos la precisión [51].

6.1.2 Validación cruzada

Para el entrenamiento de los modelos se ha usado *train_test_split* para generar aleatoriamente un conjunto de entrenamiento y otro de prueba, evitando usar todo el conjunto de datos para ambas tareas y así reducir la sobreestimación que eso conlleva.

A la hora de medir el rendimiento del modelo, el resultado obtenido puede estar ligado a esa aleatoriedad con la que se hizo la separación entre los dos conjuntos, sobre todo en *datasets* donde se presenta un alto desbalanceo entre las clases. En el caso que nos concierne, la clasificación binaria, podríamos tener el problema donde todas las muestras de la clase objetivo Normal estuviesen en el conjunto de entrenamiento y las muestras de la clase Ataque en el conjunto de prueba, dando como resultado un mal rendimiento.

Para evitar este problema se usa la validación cruzada, siendo un método estadístico para evaluar el rendimiento de generalización de un modelo creado, más fiable que la simple división del conjunto en entrenamiento y prueba.

Su funcionamiento consiste en dividir repetidamente el conjunto de datos en n partes de igual tamaño, llamados *folds*, para poder entrenar n modelos. Los *folds* se utilizarán tanto para el entrenamiento como para la validación. El proceso lo podemos ver en la figura 6.2.

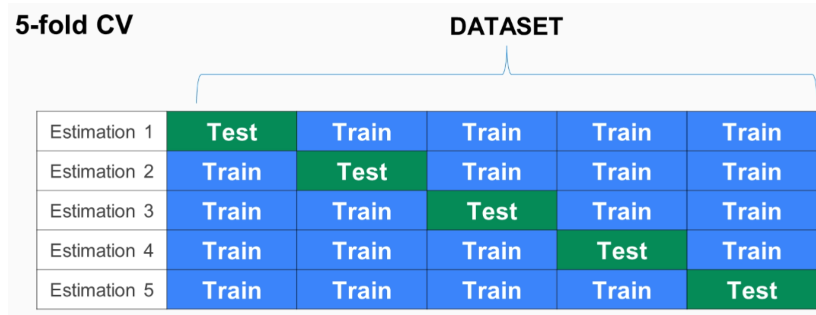


Figura 6.2: Validación cruzada [6].

Para cada conjunto modelo se calcula la precisión, dando como resultado la media de las n precisiones.

Una de las grandes desventajas de la validación cruzada es el aumento del coste computacional que supone tener que entrenar n modelos en lugar de uno, siendo este método aproximadamente n veces más lento que haciendo sólo una división.

Una buena idea puede ser utilizar la validación cruzada de *k-fold* estratificada, la cual asegura que la proporción entre las clases objetivo se mantenga igual que las del conjunto inicial en cada uno de los *folds* creados, dando como resultado estimaciones más fiables para el rendimiento de la generalización. En este caso, como anteriormente ya se realizó *undersampling*, decidiremos usar la validación cruzada sin estratificación [52].

6.1.3 Resultado de las métricas

Regresión Logística

Llegados a este punto entrenamos el modelo de Regresión Logística, y en la tabla 6.1 vemos los resultados para las métricas comentadas anteriormente, y en la figura 6.3 podemos ver la matriz de confusión resultante.

RESULTADOS DE LOS TRES MODELOS			
	REGRESIÓN LOGÍSTICA	RANDOM FOREST	NAIVE BAYES
Precisión:	0.9178	0.9992	0.8473
Exactitud:	0.9490	0.9989	0.8817
Sensibilidad:	0.9869	0.9987	0.9291
F1-score	0.9511	0.9989	0.8863
Media CV*:	0.95	1.0	0.88
Indice Kappa:	0.8981	0.9979	0.7636
Tiempo ejecución:	3.01 s	19.8 s	19 ms

* Media Validación Cruzada

Tabla 6.1: Resultados de los tres modelos

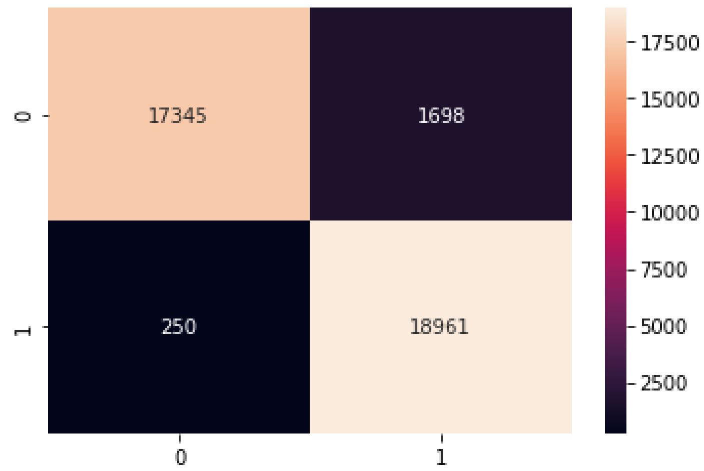


Figura 6.3: Matriz de Confusión del modelo de Regresión Logística.

Random Forest

Los resultados de las métricas del modelo de *Random Forest* los podemos ver en la tabla 6.1 y la matriz de confusión en la figura 6.4.

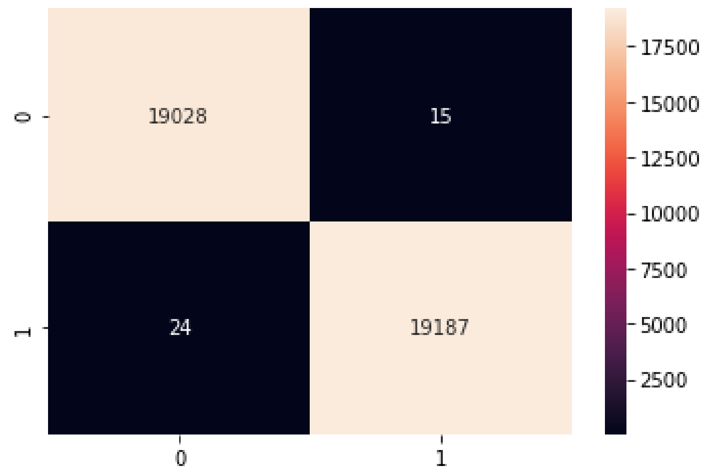


Figura 6.4: Matriz de Confusión del modelo de *Random Forest*.

Naive Bayes

Tras crear el modelo de *Naive Bayes* resultado de la combinación del modelo para los datos continuos y del modelo de los datos categóricos, los resultados finales los podemos ver en la tabla 6.1 y la matriz de confusión en la figura 6.5.

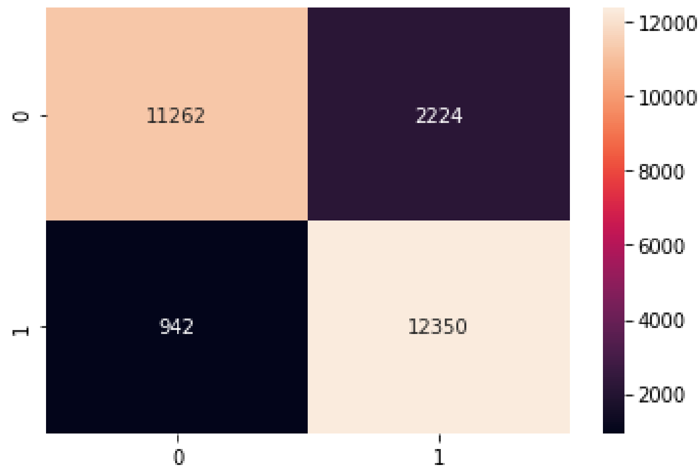


Figura 6.5: Matriz de Confusión del modelo de *Naive Bayes*.

6.2 Hiperparametrización

Esta fase consiste en ajustar los parámetros del algoritmo para intentar mejorar su comportamiento. Cabe destacar que no es lo mismo la hiperparametrización que la parametrización, puesto que esta última se refiere a los parámetros internos del algoritmo, que suelen ser coeficientes o pesos que asigna el algoritmo para cada modelo en particular, mientras que la hiperparametrización se refiere a los parámetros que pueden ser ajustados por parte del usuario [53].

Entrando en el API de la librería podemos ver la lista de parámetros que se pueden modificar, pero llegar a saber cuáles cambiar para mejorar el modelo puede resultar bastante complejo. Es por eso por lo que se suele emplear alguna estrategia, como puede ser la denominada *Grid Search*.

Grid Search es el proceso de escanear los datos para configurar los parámetros de un modelo, lo cual en función del tipo de modelo se requieren ciertos parámetros.

Esta técnica lo que hace es crear un modelo con cada combinación de parámetros posible, lo cual puede resultar una tarea computacionalmente muy compleja, en función del número de posibles combinaciones de parámetros y del tamaño del conjunto de datos. El resultado de este proceso es la combinación de parámetros con la que el modelo es más preciso [54].

Por lo tanto, para cada uno de los algoritmos intentaremos ejecutarlos con la mejor combinación de parámetros resultante de aplicar *Grid Search*, comparando así estos resultados con los obtenidos en las secciones anteriores, en las que se usaron los parámetros por defecto.

RESULTADOS DE LOS TRES MODELOS HIPERPARAMETRIZADOS			
	REGRESIÓN LOGÍSTICA	RANDOM FOREST	NAIVE BAYES**
Precisión:	0.9350	0.9991	0.8473
Exactitud:	0.9593	0.9988	0.8817
Sensibilidad:	0.9876	0.9985	0.9291
F1-score	0.9606	0.9988	0.8863
Media CV*:	0.96	1.0	0.88
Indice Kappa:	0.9186	0.9977	0.7636
Tiempo ejecución:	9.18 s	19.6 s	19 ms

* Media Validación Cruzada
** Naive Bayes sin hiperparametrizar

Tabla 6.2: Resultados de los tres modelos hiperparametrizados

6.2.1 Regresión Logística

El algoritmo Regresión Logística no tiene un hiperparámetro crítico que ajustar, y debido a que los resultados ya son muy buenos podríamos obviar este paso, pero por hacer una prueba, aplicaremos *Grid Search* para ver qué valores serían los más correctos para los siguientes parámetros:

- **solver:** algoritmo a usar para la optimización del problema, pudiendo ser útil para el rendimiento o la convergencia.
- **penalty:** se utiliza para especificar la norma utilizada para la penalización.
- **C:** controla la intensidad de la penalización.

Con los valores **C**: 100, **penalty**: l2, **solver**: liblinear vemos en la tabla 6.2 que obtenemos una leve mejora en la mayoría de las métricas, teniendo como coste que el tiempo de ejecución aumenta un poco.

6.2.2 *Random Forest*

Para *Random Forest*, aplicar *Grid Search* puede que no tenga mucho sentido, puesto que el resultado de la validación cruzada del modelo con los valores que habíamos usado ya daba 1.0, pero aun así haremos un breve análisis de algunos de los parámetros que podríamos ajustar en caso de ser necesario y haremos una pequeña prueba de *Grid Search* para ver el comportamiento:

- **max_features:** es el parámetro más importante, marcando el número de características aleatorias a muestrear para buscar la mejor división. Se puede usar un número o una cadena de caracteres como 'sqrt', lo cual indica que el número de características a

muestrear viene dado por el resultado de la raíz cuadrada del número total de variables del conjunto, siendo esta una de las más usadas.

- **n_estimators:** indica el número de árboles.
- **max_depth:** indica la profundidad máxima del árbol. Puede ser útil para evitar que el árbol se extienda hasta que todas las hojas sean puras, reduciendo de esta forma el tiempo de ejecución.

Los resultados de aplicar *Grid Search* son: **max_features**='sqrt', **n_estimators**=200 y **max_depth**=90, dando unas métricas casi idénticas a las obtenidas con los parámetros utilizados anteriormente, como vemos en la tabla 6.2.

6.2.3 *Naive Bayes*

Según lo hecho anteriormente, lo ideal sería aplicar *Grid Search* sobre el modelo de GaussianNB y BernoulliNB por separado, pero nos encontramos con la limitación de que el primero solo posee dos parámetros para ajustar (**priors:** probabilidades previas de las clases, y **var_smoothing:** porción de la mayor variación de todas las características que se agrega a las variaciones para la estabilidad del cálculo), para los cuales resulta algo complejo encontrar posibles combinaciones de valores que puedan mejorar los resultados, mientras que para el segundo algoritmo, BernoulliNB, el número de parámetros ya crece hasta cuatro, pero nos volvemos a encontrar con la misma situación, donde el parámetro que más nos interesaría sería **binarize**, el cual consiste en un umbral para binarizar las características, pero en nuestro caso no nos hace falta puesto que el conjunto de datos que se utiliza en este modelo ya está binarizado desde la fase de preparación de los datos.

Por lo tanto, en el caso de *Naive Bayes* prescindiremos de la hiperparametrización.

Capítulo 7

Evaluación

Llegados a esta fase es el momento de evaluar y comparar los tres modelos creados. Destacar que si es necesario se puede volver a algún paso anterior para revisar y corregir algún problema y así mejorar los modelos. Es por este motivo por lo que en la primera iteración en la que se obtuvieron los resultados de rendimiento de los primeros modelos para probar las características iniciales y obtener una breve idea de su funcionamiento, apreciamos que estos ya arrojaban unos resultados bastante buenos para lo poco desarrollados que estaban los modelos, sobre todo teniendo en cuenta las características que estábamos utilizando. En especial, *Random Forest* entrenado con sólo las características **totalSourceBytes**,**totalDestinationBytes**,**totalSourcePackets**,**totalDestinationPackets**,**sourcePort** y **destinationPort** obteníamos unos resultados casi perfectos, lo que nos hicieron dudar de si se habían seguido los pasos correctos de la metodología y si se estaban tratando correctamente los datos. Tras volver a los pasos previos y realizar sucesivas pruebas y revisiones en cada fase, no apreciamos que se estuviese cometiendo ningún error, y comparado con los otros dos algoritmos, vimos que los resultados continuaban siendo bastante buenos pero que siguiendo las pautas marcadas por la metodología aún había un pequeño margen de mejora, por lo que continuamos con el trabajo, a pesar de que esto supuso un breve retraso en la planificación. Estos resultados comentamos los podemos ver en la tabla 7.1

Otra posible opción sería cambiar el conjunto de datos, puesto que la razón que dimos para los buenos resultados era que el escenario sintético en el que se generó el tráfico no fuese lo suficientemente bueno (a pesar de ser ampliamente usado) y propiciase que las clases fuesen fácilmente separables ante los algoritmos de *Machine Learning*, puesto que analizando los datos no se apreciaba dicha situación.

Haciendo una breve prueba con otro *dataset* más reciente de la misma fuente del que se obtuvo este, vimos que nuestra hipótesis se confirmaba al obtener una situación muy parecida, por lo que decidimos continuar para adelante.

Dicho lo anterior, pasamos a analizar los resultados de los modelos finales con todas las

RESULTADOS DE LOS TRES MODELOS TEMPRANOS			
	REGRESIÓN LOGÍSTICA	RANDOM FOREST	NAIVE BAYES
Precisión:	0.9297	0.9976	0.8396
Exactitud:	0.9418	0.9978	0.8967
Sensibilidad:	0.9554	0.9980	0.9795
F1-score	0.9423	0.9978	0.9042
Media CV*:	0.94	1.0	0.90
Indice Kappa:	0.8837	0.9956	0.7935

* Media Validación Cruzada

Tabla 7.1: Resultados de los tres modelos tempranos

fases de la metodología realizadas. El modelo que más destaca es el de *Random Forest* por sus llamativos resultados, rondando casi todas las métricas el uno. Esto se debe a lo comentado en el anterior párrafo, pero también concuerda con la explicación hecha sobre el algoritmo, ya que como *Random Forest* es un algoritmo robusto y potente esperábamos buenos resultados sin sobreajuste.

También debido a su alta complejidad computacional, es el algoritmo cuyo tiempo de entrenamiento es mayor.

Como segundo modelo que mejores resultados arroja podemos decir que es Regresión Logística donde ya nos esperábamos que los resultados fuesen algo peores que los de *Random Forest* al tener una pequeña tendencia al sobreajuste, aunque eso sí, su tiempo de entrenamiento es menor.

Finalmente tenemos a *Naive Bayes*, el cual cumple todas nuestras expectativas. Es el que peor resultados arroja, aunque de todas formas son bastante buenos, confirmando que la combinación de los modelos de GaussianNB y BernoulliNB han funcionado. Otro aspecto esperado y que se ha confirmado es que el tiempo de ejecución de *Naive Bayes* es el más rápido de todos los modelos.

Comparando los resultados finales con sus predecesores también vemos que los resultados van mejorando, por lo que los pasos realizados en cada fase del trabajo han sido los correctos y han funcionado.

A modo resumen, el algoritmo que mejores resultados arroja es *Random Forest*, siendo el que más tiempo se toma, mientras que *Naive Bayes* es el que peor resultados obtiene a cambio de un tiempo de entrenamiento muy inferior.

Capítulo 8

Despliegue

En esta fase lo normal sería hacer el despliegue de los modelos creados en las anteriores fases, pero debido a que uno de los objetivos es intentar hacer el despliegue en un sistema distribuido, tenemos que usar tecnologías distintas a las anteriores, lo que conlleva volver a crear los modelos.

En concreto, implementamos los modelos en la plataforma Databricks, haciendo uso de la librería MLlib de Apache Spark y de PySpark.

En un previo estudio sobre las tecnologías a utilizar para esta fase, la idea principal era montar un entorno, ya fuese en AWS o en Azure, en el que tuviésemos un *cluster* con Spark para ejecutar los modelos y otro *cluster* con Apache Kafka que sirviese como ingestador de datos a los modelos. En este estudio vimos algunos problemas, que comentaremos posteriormente, pero uno de los principales era el tiempo que suponía desplegar y terminar los *clusters*, excesivo para realizar las diferentes pruebas de rendimiento con los tres algoritmos.

Continuando con el estudio de tecnologías, apreciamos que con Databricks el tiempo de despliegue de los *clusters* era siempre inferior a los cinco minutos, por lo que este fue uno de los motivos por los que se optó por usar esta tecnología.

Databricks ofrece una versión gratuita limitada a un *cluster*, pero para realizar distintas pruebas de rendimiento tuvimos que elegir uno de los planes de pago y la infraestructura a utilizar: AWS o Microsoft Azure. En un primer momento se escogió Azure como infraestructura y se implementaron los modelos en ella.

Para comenzar a trabajar, primero hay que crear un *cluster*, teniendo que elegir el tipo de controlador y el tipo y cantidad de trabajadores. A partir de aquí ya podremos trabajar como en las fases anteriores, puesto que Databricks permite trabajar de forma interactiva mediante *notebooks*, pudiendo escribir y ejecutar el código por celdas.

Un prerrequisito importante, como ya se ha comentado anteriormente, es ver si los algoritmos que queremos usar están implementados en Spark MLlib. En este caso ya se había comprobado al inicio del proyecto.

El siguiente paso es disponer del *dataset* que hemos utilizado anteriormente. Para esto debemos cargar los datos en el sistema, una tarea muy sencilla gracias al uso de la interfaz web que proporciona la plataforma, con la que podemos cargar los datos en el DBFS (*Databicks File System*), un sistema de archivos distribuido que se monta en el espacio de trabajo y que está disponible para los *clusters*.

Cargado el conjunto de datos, comenzamos a crear los tres modelos con los mejores parámetros y características seleccionadas en las anteriores etapas. En este paso hemos intentado hacer los modelos lo más parecidos posible a los implementados con Scikit-Learn, puesto que MLlib no dispone exactamente de los mismos hiperparámetros.

Tras la implementación, entrenamiento y validación podemos comparar los resultados con los obtenidos con la librería Scikit-Learn. Pero antes de eso, un aspecto a destacar es la forma de implementar el modelo de *Naive Bayes*. Al igual que en la sección 5.3.3, nos encontramos con el problema de que lo más normal es tratar los datos continuos y discretos por separado. El problema encontrado con el uso de MLlib es que no se puede aplicar la misma solución que en la sección 5.3.3. Por un lado, no podemos obtener las probabilidades del modelo resultado de aplicar Bernoulli, y por otro lado no hay la implementación de Gaussian, hay la Multinomial que se suele utilizar para los datos continuos, pero debido a estos dos impedimentos, se ha optado por utilizar solo los datos discretos para crear el modelo de Bernoulli, y así comparar los resultados de esta fase con los obtenidos en el modelo intermedio de la solución implementada utilizando Scikit-Learn.

En la tabla 8.1 podemos comparar los resultados de ejecutar la validación cruzada en los modelos finales de Scikit-Learn y Spark. Resaltar que la métrica utilizada en ambos es el *Area Under Receiver Operating Characteristic Curve* (ROC AUC), puesto que era la única que estaba implementada en ambas librerías para la clasificación binaria [55][56] .

RESULTADOS VALIDACIÓN CRUZADA		
	Scikit-Learn	Spark
Regresión Logística	0.98	0.94
Random Forest	1.0	0.978
Naive Bayes(Benoulli)	0.98	0.86

Tabla 8.1: Comparación de resultados entre Scikit-Learn y Spark.

Podemos ver que Spark arroja unos resultados un poco peores. Esto en parte es lógico debido a que con la paralelización se pierde algo de precisión, y los parámetros utilizados no pudieron ser exactamente los mismos, como fue en el caso de *Random Forest*, por ejemplo, donde con Scikit-Learn el parámetro referente a la profundidad no lo podemos establecer a *None* (hasta que alcance el final) mientras que en Spark la profundidad máxima admitida es de treinta.

8.1 Pruebas de rendimiento

Otro de los objetivos de esta última fase es analizar y comparar el rendimiento de los tres modelos tras realizar la implementación distribuida.

Usando el *dataset* original (tras aplicar *undersampling*), el mismo que ya se utilizó para obtener las métricas, nos encontramos con un conjunto de datos insuficiente para poder analizar detalladamente el rendimiento, por lo que procedemos a crear un *dataset* mucho más grande originado de la concatenación repetida del *dataset* original, obteniendo un conjunto de unos 12 millones de muestras, y así tener la cantidad suficiente como para simular un escenario real con un flujo constante de datos.

Cargamos los modelos guardados anteriormente y procedemos a validar el modelo con los datos ingestados en ”tiempo real”, utilizando el propio *cluster* como ingestador mediante la funcionalidad de *Structured Streaming* de Spark, por lo que crearemos una tabla donde mostraremos los ratios de Verdaderos Positivos y Verdaderos Negativos, que se irá actualizando según se vayan leyendo nuevos datos [57].

Es preciso mencionar que la idea principal era utilizar la plataforma Azure para el despliegue de un *cluster* con Apache Spark o mediante Databricks, y también Apache Kafka como ingestador. El uso de Kafka requiere que se encuentre dentro de la misma red privada que sus consumidores, lo cual en un principio no suponía ningún problema puesto que la creación de un *cluster* dentro de una red privada en Azure no implica ninguna complejidad. El problema se encontró a la hora de cargar los flujos de datos en Kafka, puesto que el productor ejecutado desde mi ordenador también debía estar en la misma red privada, lo cual solo se puede hacer mediante el uso de una VPN (*Virtual Private Network*).

El problema de usar la VPN no era montar la parte cliente, lo cual sería lo más fácil, sino que la VPN la hay que montar también desde el lado servidor, desde dentro de la red privada, siendo un proceso no tan trivial, que a pesar de existir documentación al respecto, es complejo y el tiempo empleado excedería de lo estimado. Es por este motivo por lo que se decidió buscar otra alternativa, y usar la propia funcionalidad *Structured Streaming* de Spark.

Continuando con *Structured Streaming*, una de las ventajas que presenta es que sin tener que escribir ningún código adicional, nos permite visualizar las métricas de *Input Rate*, *Processing Rate* y *Batch Duration*.

El *Input Rate* especifica la cantidad de datos que fluyen hasta el *Structured Streaming* desde el sistema ingestador, en este caso el propio *cluster*.

El *Processing Rate* especifica la rapidez con la que se pueden analizar esos datos del flujo.

Lo normal es que estos datos varíen de forma conjunta, pero se puede dar la situación en la que el *Processing Rate* sea muy inferior al *Input Rate*, siendo síntoma de que hay que reescalar el *cluster* añadiendo más trabajadores.

El *Batch Duration* se utiliza para medir el rendimiento del procesamiento por lotes que suelen usar los sistemas de transmisión, siendo un comportamiento normal que cuantos más trabajadores tenga el *cluster* menor sea el *Batch Duration* (mayor rendimiento).

Un ejemplo de estas métricas lo podemos ver en la figura 8.1.

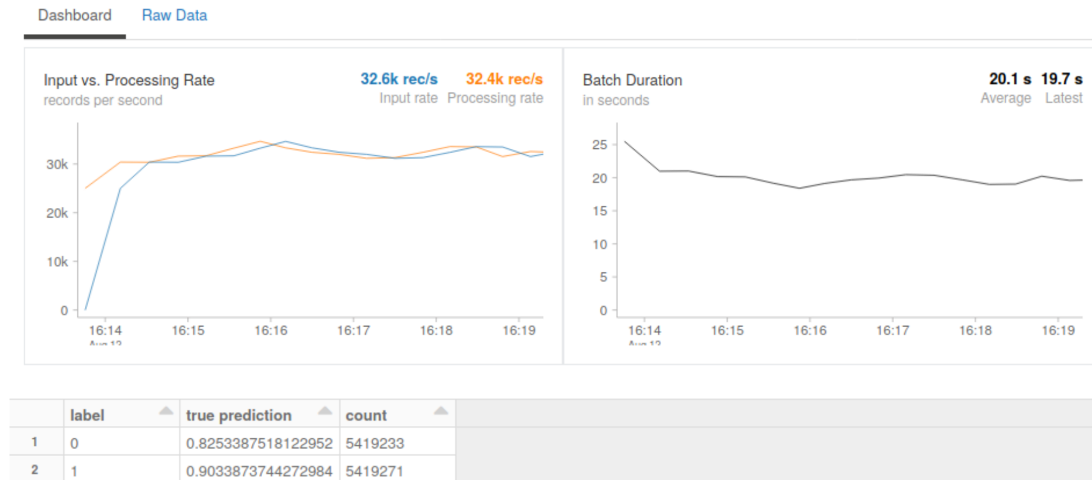


Figura 8.1: Métricas de rendimiento para *Naive Bayes* distribuido con dos trabajadores.

Las pruebas realizas han sido con los tres algoritmos, ejecutando cada uno en un *cluster* con dos, cuatro y ocho trabajadores. En las tablas 8.2, 8.3, 8.4 podemos ver los resultados de estas métricas para los modelos Regresión Logística, *Random Forest* y *Naive Bayes* (Bernoulli), respectivamente.

MÉTRICAS REGRESIÓN LOGÍSTICA			
NºCusters	Input Rate (rec/s)	Processing Rate (rec/s)	Batch Duration
2	30.7k	30.7k	21.8s
4	55.8k	56.6k	12.5s
8	88.2k	67.5k	8.2s

Tabla 8.2: Métricas de Regresión Logística.

MÉTRICAS RANDOM FOREST			
NºCusters	Input Rate (rec/s)	Processing Rate (rec/s)	Batch Duration
2	10.2k	10.2k	1 min
4	21.2k	21.4k	29s
8	37.4k	36.7k	17.3s

Tabla 8.3: Métricas de *Random Forest*.

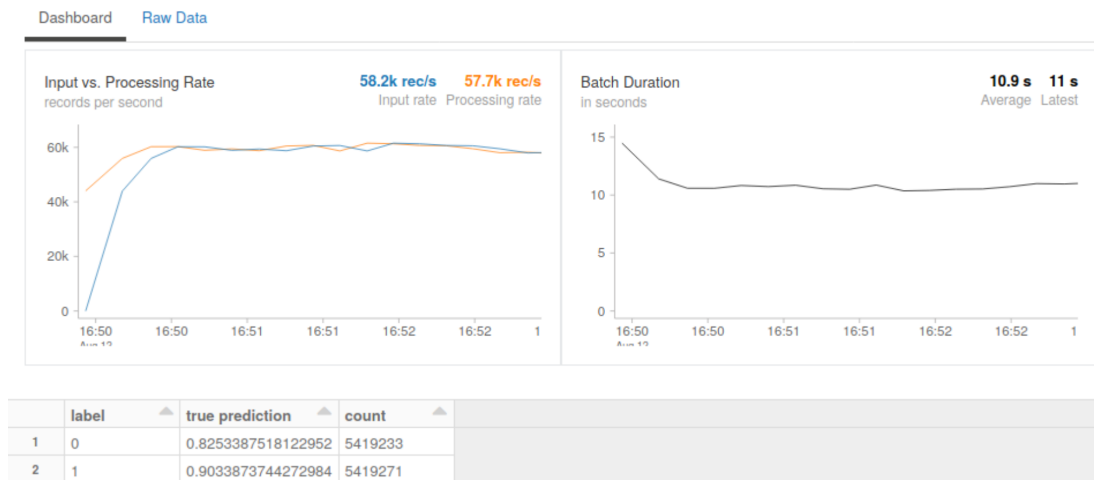
En ellas apreciamos que según se aumenta el número de trabajadores el *Batch Duration* disminuye, mientras que el *Input* y *Processing Rate* aumentan por igual, teniendo por lo tanto

MÉTRICAS NAIVE BAYES			
NºCusters	Input Rate (rec/s)	Processing Rate (rec/s)	Batch Duration
2	32.6k	32.4k	20.1s
4	58.2k	57.7k	10.9s
8	98k	96.3k	6.5s

Tabla 8.4: Métricas de *Naive Bayes* (Bernoulli).

un buen rendimiento según lo visto de forma teórica.

En las figuras 8.1, 8.2 y 8.3, podemos ver este comportamiento de manera gráfica para el algoritmo *Naive Bayes*. Para Regresión Logística y *Random Forest* las gráficas serían casi idénticas salvo que con valores diferentes.



	label	true prediction	count
1	0	0.8253387518122952	5419233
2	1	0.9033873744272984	5419271

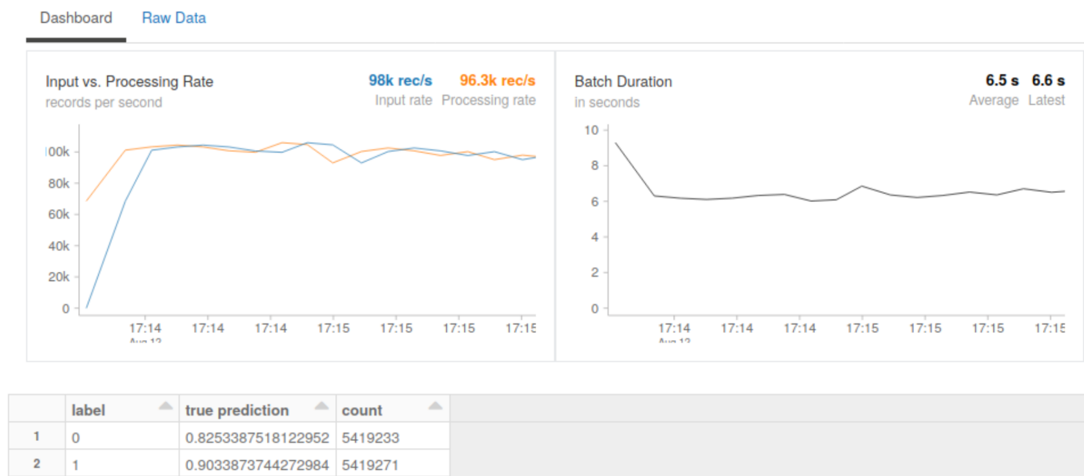
Figura 8.2: Métricas de rendimiento para *Naive Bayes* distribuido con cuatro trabajadores.

Destacar que para realizar estas pruebas, tuvimos que migrar los modelos de Databricks de la infraestructura de Microsoft Azure a AWS, puesto que en Azure tuvimos problemas con las cuotas de CPU al intentar aumentar el número de nodos trabajadores, mientras que en AWS no encontramos dichas restricciones. Esta migración ocasionó un breve retraso en la planificación.

También es interesante monitorizar el *cluster* y ver las métricas de rendimiento de los trabajadores, tanto en conjunto como por separado. Para esto existen diversas herramientas, como pueden ser Zabbix, Cacti, Nagios, etc.

Finalmente se optó por utilizar Ganglia, una herramienta de monitorización distribuida y escalable, que ya viene integrada con Databricks, pudiendo acceder a su interfaz sin necesidad de instalar nada adicionalmente, lo cual ahorra bastante tiempo.

Para una mejor experiencia en la monitorización, se configuró la variable de entorno **DATABRICKS_GANGLIA_SNAPSHOT_PERIOD_MINUTES** a un minuto, para que Data-



	label	true prediction	count
1	0	0.8253387518122952	5419233
2	1	0.9033873744272984	5419271

Figura 8.3: Métricas de rendimiento para *Naive Bayes* distribuido con ocho trabajadores.

bricks recolecte las métricas de Ganglia con esa frecuencia.

Tras ejecutar todas las métricas y visualizar toda la información monitorizada, lo primero que se observa es que según aumentamos el número de nodos trabajadores el tiempo de ejecución disminuye, según lo esperado.

También se aprecia como el tráfico de red de entrada aumenta según aumentan los nodos. Esto también es lo esperado según lo visto con las métricas anteriores de *Input Rate* y *Processing Rate*, donde al aumentar la cantidad de nodos también aumentaba la capacidad del *cluster*.

Este aumento de capacidad también se debe a un mayor uso de la memoria, mientras que la media de uso de CPU disminuye.

En cuanto a las gráficas referentes a la carga de cada nodo del *cluster*, se puede observar un claro desbalanceo, donde el mayor equilibrio se da con el modelo de *Random Forest*, pudiendo ser a causa de su mayor complejidad computacional en comparación con los otros dos modelos.

En las figuras 8.4, 8.5 y 8.6 podemos ver estas métricas comentadas y el rendimiento en el caso concreto para *Random Forest*, utilizando dos, cuatro y ocho nodos trabajadores, respectivamente.

En las figuras 8.7, 8.8 y 8.9 podemos ver como se distribuye la carga de trabajo entre los distintos nodos del *cluster* para *Random Forest* con dos, cuatro y ocho nodos trabajadores, respectivamente.

Como se comentó en la sección 1.5, en el repositorio están todas las gráficas resultantes de las pruebas realizadas en esta parte del proyecto.

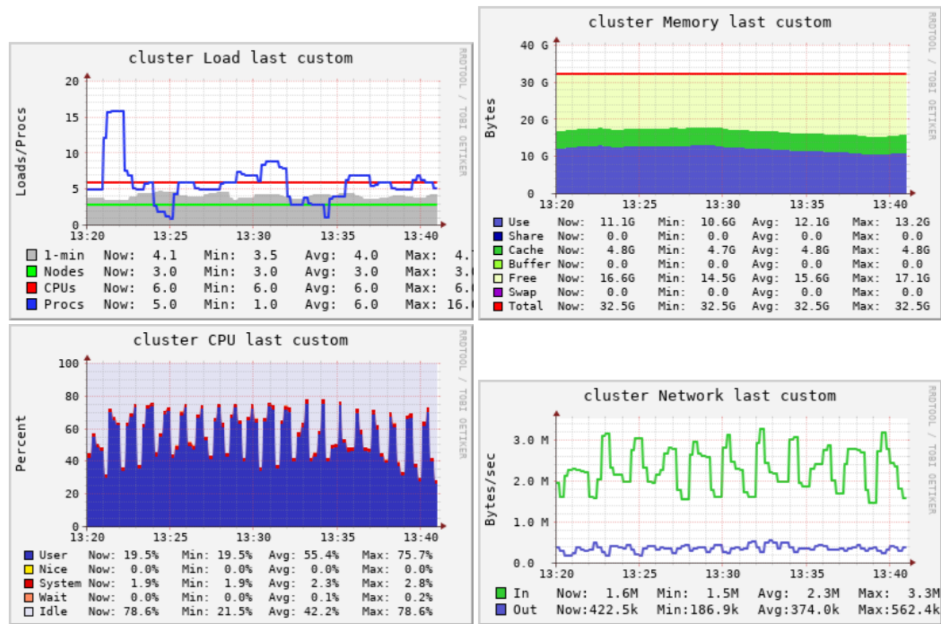


Figura 8.4: Métricas de rendimiento del *cluster* para *Random Forest* distribuido con dos trabajadores.

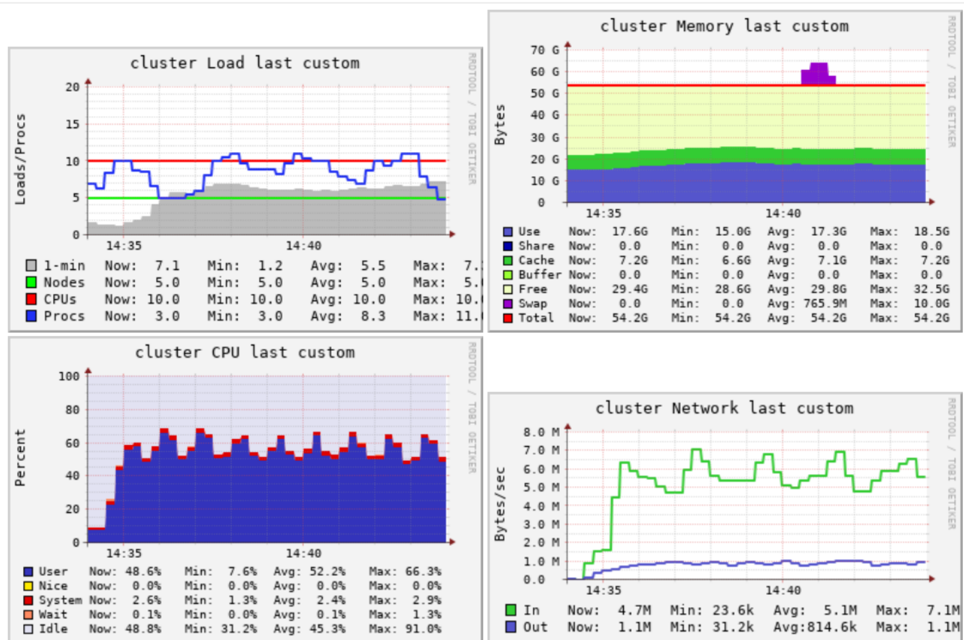


Figura 8.5: Métricas de rendimiento del *cluster* para *Random Forest* distribuido con cuatro trabajadores.

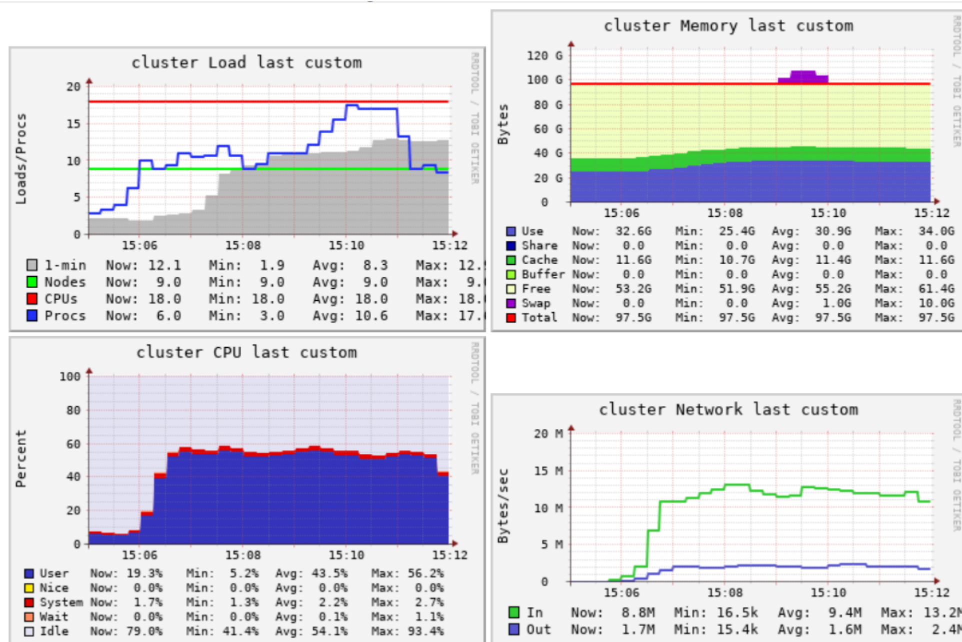


Figura 8.6: Métricas de rendimiento del *cluster* para *Random Forest* distribuido con ocho trabajadores.

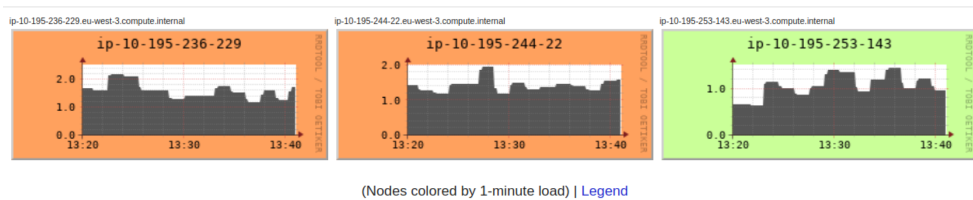


Figura 8.7: Distribución de la carga de trabajo del *cluster* para *Random Forest* distribuido con dos nodos trabajadores.

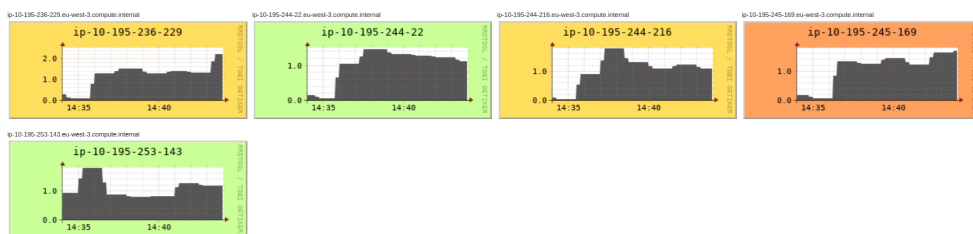


Figura 8.8: Distribución de la carga de trabajo del *cluster* para *Random Forest* distribuido con cuatro nodos trabajadores.

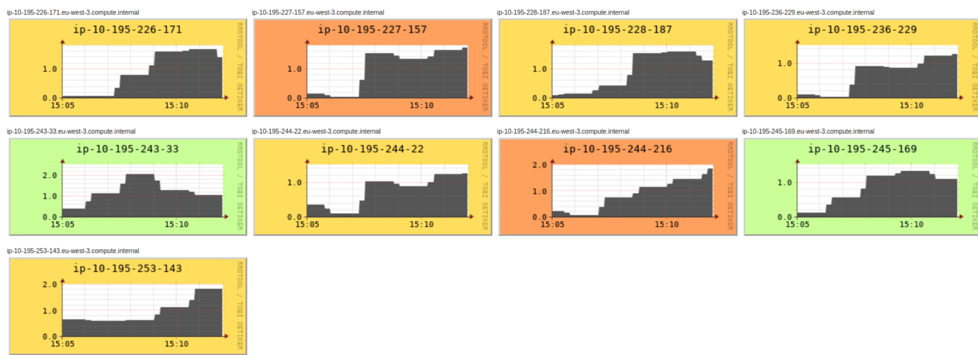


Figura 8.9: Distribución de la carga de trabajo del *cluster* para *Random Forest* distribuido con ocho nodos trabajadores.

Capítulo 9

Conclusiones y líneas futuras

Este trabajo ha supuesto para mí una oportunidad de poner en práctica los conceptos aprendidos y habilidades adquiridas durante los años de estudio del Grado en Ingeniería Informática para poder afrontar el reto de realizar de forma autónoma un proyecto como este.

Analizando el grado de cumplimiento de los objetivos, a pesar de que en algunas fases se ha encontrado alguna dificultad, podemos afirmar que todos los objetivos marcados se han logrado. Revisando estas metas, una muy importante era la aplicación de ingeniería de características sobre el *dataset*, la cual se ha logrado por medio de la limpieza, transformación y selección de las variables existentes.

Otro objetivo era el estudio de diferentes técnicas de aprendizaje máquina utilizadas en la clasificación de tráfico de red. Esto se cumplió al hacer un estudio pormenorizado del estado del arte. Se seleccionaron tres algoritmos en base a sus resultados en trabajos previos y teniendo en cuenta su distinta naturaleza, lo que enriquece el trabajo realizado.

Con los algoritmos seleccionados, se crearon tres modelos que fueron validados y evaluados usando diferentes métricas. Asimismo, empleando la librería MLlib se implementaron los modelos para desplegarlos en un entorno distribuido utilizando la plataforma Databricks. Además de ser evaluados con las métricas de precisión en la clasificación, se analizó el rendimiento de los algoritmos por medio de la información obtenida con herramientas de monitorización.

En mi opinión los primeros pasos dados fueron los más difíciles, debido a tener que gestionar y planificar un proyecto de estas dimensiones sin una experiencia previa. Al avanzar en el desarrollo fui adquiriendo ciertas destrezas según iba logrando los objetivos gracias a las pautas que se habían marcado. Además, los contratiempos que fueron apareciendo, sin lugar a dudas, me aportaron una experiencia muy valiosa.

Por otro lado, la temática de este trabajo me permitió descubrir y conocer más en detalle aspectos relacionados con *Machine Learning* y *Big Data*.

En este trabajo se han asumido diferentes perfiles. Uno de ellos, el más cercano a mi itinerario, ha sido el orientado a ingeniería en la parte del proyecto en la que se realiza el estudio

sobre los flujos de red y en la que se aborda el uso de sistemas distribuidos.

Otro perfil ha sido el de científico de datos. El trabajo realizado me ha permitido profundizar en esta área de conocimiento, descubriendo la gran comunidad que la ampara.

En cuanto a posibles líneas futuras, destacaría las siguientes:

- Profundizar más en la optimización del código de creación de las características, intentado reducir aún más el tiempo de ejecución.
- Realizar el despliegue de los algoritmos seleccionados sobre Apache Spark utilizando Apache Kafka como ingestador de datos.
- Probar todo el proceso de ingeniería de características de forma distribuida.
- Incorporar algoritmos de *Deep Learning* al análisis comparativo.

Lista de acrónimos

CRISP-DM *Cross Industry Standard Process for Data Mining*. 7

CSV *Comma-Separated Values*. 30

DAG *Directed Acyclic Graph*. 20

DBFS *Databicks File System*. 60

DBSCAN *Density-based Spatial Clustering of Applications with Noise*. 35

IDS Sistemas de Detección de Intrusos. 1

IPS Sistemas de Prevención de Instrusos. 1

KDD *Knowledge Discovery in Databases*. 7

ML *Machine Learning*. 1

PCA *Principal Component Analysis*. 47

RDD *Resilient Distributed Datasets*. 20

RFECV *Recursive Feature Elimination with Cross-Validation*. 43

SEMMA *Sample, Explore, Modify, Model, Assess*. 7

TCP *Transmission Control Protocol*. 32

VPN *Virtual Private Network*. 61

XML *eXtensible Markup Language*. 30

Bibliografía

[1] Datahack, "Mllib, la biblioteca de machine learning de spark." [En línea]. Disponible en: https://www.datahack.es/mllib-machine-learning-spark/

[2] V. Chavez, "Arboles decision." [En línea]. Disponible en: https://rpubs.com/elfenixsoy/arbol-veronica

[3] R. Gandhi, "Naive bayes classifier." [En línea]. Disponible en: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

[4] Databricks, "Apache spark." [En línea]. Disponible en: https://databricks.com/spark/about

[5] A. R. Mesa, "¿qué es apache spark?" [En línea]. Disponible en: https://openwebinars.net/blog/que-es-apache-spark/

[6] [En línea]. Disponible en: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789617740/2/ch02lvl1sec14/k-fold-cross-validation

[7] Domo, "Data never sleeps 7.0." [En línea]. Disponible en: https://www.domo.com/learn/data-never-sleeps-7

[8] PowerData, "Big data: ¿en qué consiste? su importancia, desafíos y gobernabilidad." [En línea]. Disponible en: https://www.powerdata.es/big-data

[9] A. D'Alconzo, I. Drago, A. Morichetta, M. Mellia, and P. Casas, "A survey on big data for network trafficmonitoring and analysis." [En línea]. Disponible en: https://ieeexplore.ieee.org/document/8789667

[10] R. Hofstede, P. Celeda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and ipfix." [En línea]. Disponible en: https://ieeexplore.ieee.org/document/6814316

[11] ECURED, "Computación distribuida." [En línea]. Disponible en: https://www.ecured.cu/Computaci%C3%B3n_distribuida

[12] Bogotobogo, "Web technologies - distributed computing / big data 2020." [En línea]. Disponible en: https://www.bogotobogo.com/WebTechnologies/distributedcomputing.php

[13] S. S. Gupta. Real-time big data analytics. [En línea]. Disponible en: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784391409/1/ch01lvl1sec13/distributed-batch-processing

[14] E. L. Guzmán, "Módulo minería de datos." [En línea]. Disponible en: https://disi.unal.edu.co/~eleonguz/cursos/md/presentaciones/Sesion5_Metodologias.pdf

[15] R. M. M. Teresa, Álvarez Cabal J. Valeriano, M. F. J. Manuel, and G. V. Adolfo, "MetodologÍas para la realizaciÓn de proyectos de data mining." [En línea]. Disponible en: https://www.aeipro.com/files/congresos/2003pamplona/ciip03_0257_0265.2134.pdf

[16] A. Müller and S. Guido, *Introduction to Machine Learning with Python*, 1st ed. O'Reilly, 2016.

[17] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 1st ed. O'Reilly, 2017.

[18] P. R. de los Santos, "Tipos de aprendizaje en machine learning: supervisado y no supervisado." [En línea]. Disponible en: https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/

[19] V. Román, "Machine learning supervisado: Fundamentos de la regresión lineal." [En línea]. Disponible en: https://medium.com/datos-y-ciencia/machine-learning-supervisado-fundamentos-de-la-regresi%C3%B3n-lineal-bbcb07fe7fd/

[20] J. I. Bagnato, "Arbol de decisión en python: Clasificación y predicción." [En línea]. Disponible en: https://www.aprendemachinelearning.com/arbol-de-decision-en-python-clasificacion-y-prediccion/

[21] V. Roman, "Algoritmos naive bayes: Fundamentos e implementación." [En línea]. Disponible en: https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f

[22] L. González, "Naive bayes – teoría." [En línea]. Disponible en: https://ligdigonzalez.com/naive-bayes-teoria-machine-learning/

[23] U. I. de Valencia, "Machine learning python: el lenguaje de los negocios del futuro." [En línea]. Disponible en: https://www.universidadviu.es/machine-learning-python-el-lenguaje-de-los-negocios-del-futuro/

[24] GangBoard, "What is pyspark?" [En línea]. Disponible en: https://www.gangboard.com/blog/what-is-pyspark

[25] Databricks, "Pyspark." [En línea]. Disponible en: https://databricks.com/glossary/pyspark

[26] L. González, "Introducción a la librería scikit-learn de python." [En línea]. Disponible en: https://ligdigonzalez.com/libreria-scikit-learn-de-python/

[27] Óscar Fernández, "Databricks: Introducción a spark en la nube." [En línea]. Disponible en: https://aprenderbigdata.com/databricks/

[28] A. D. Martín, "¿qué es azure databricks?" [En línea]. Disponible en: https://blogs.encamina.com/por-una-nube-sostenible/que-es-azure-databricks/

[29] A. Kafka, "Introduction: Everything you need to know about kafka in 10 minutes." [En línea]. Disponible en: https://kafka.apache.org/intro

[30] V. Madrid, "Aprendiendo apache kafka (parte 3) : Conceptos básicos para desarrollo." [En línea]. Disponible en: https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-3-conceptos-basicos-extra/

[31] E. Abellán, "Metodología scrum: qué es y cómo funciona." [En línea]. Disponible en: https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html

[32] K. Schwaber and J. Sutherland, "La guía de scrum." [En línea]. Disponible en: https://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100

[33] M. Rehkopf, "Épicas, historias, temas e iniciativas." [En línea]. Disponible en: https://www.atlassian.com/es/agile/project-management/epics-stories-themes

[34] "Convenio colectivo estatal de empresas de consultoría, y estudios de mercados y de la opinión pública," 2018. [En línea]. Disponible en: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf

[35] C. I. for Cybersecurity, "Intrusion detection evaluation dataset (iscxids2012)." [En línea]. Disponible en: https://www.unb.ca/cic/datasets/ids.html

[36] S. McLeod, "What does a box plot tell you?" [En línea]. Disponible en: https://www.simplypsychology.org/boxplots.html

[37] W. Badr, "5 ways to detect outliers/anomalies that every data scientist should know (python code)." [En línea]. Disponible en: https://towardsdatascience.com/5-ways-to-detect-outliers-that-every-data-scientist-should-know-python-code-70a54335a623

[38] C. Jose, "Anomaly detection techniques in python." [En línea]. Disponible en: https://medium.com/learningdatascience/anomaly-detection-techniques-in-python-50f650c75aaf

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

[40] A. Shiravi, H. Shiravi, M. Tavallaee, and A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection." [En línea]. Disponible en: https://www.sciencedirect.com/science/article/pii/S0167404811001672

[41] J. Brownlee, "Random oversampling and undersampling for imbalanced classification." [En línea]. Disponible en: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

[42] S. Asaithambi, "Why,how and when to scale your features." [En línea]. Disponible en: https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e

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

[44] R. Shaikh, "Feature selection techniques in machine learning with python." [En línea]. Disponible en: https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f3

[45] J. Brownlee, "How to choose a feature selection method for machine learning." [En línea]. Disponible en: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

[46] J. I. B. Arce, "La matriz de confusión y sus métricas." [En línea]. Disponible en: https://www.juanbarrios.com/matriz-de-confusion-y-sus-metricas/

[47] P. R. de los Santos, "Machine learning a tu alcance: La matriz de confusión." [En línea]. Disponible en: https://empresas.blogthinkbig.com/ml-a-tu-alcance-matriz-confusion/

[48] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, 1st ed. O'Reilly, 2017.

[49] B. Shmueli, "Multi-class metrics made simple, part iii: the kappa score (aka cohen's kappa coefficient)." [En línea]. Disponible en: https://towardsdatascience.com/multi-class-metrics-made-simple-the-kappa-score-aka-cohens-kappa-coefficient-bdea137af09c

[50] K. Pykes, "Cohen's kappa." [En línea]. Disponible en: https://towardsdatascience.com/cohens-kappa-9786ceceab58

[51] V. Gandhi, "How to choose right metric for evaluating ml model." [En línea]. Disponible en: https://www.kaggle.com/vipulgandhi/how-to-choose-right-metric-for-evaluating-ml-model

[52] A. Müller and S. Guido, *Introduction to Machine Learning with Python*, 1st ed. O'Reilly, 2016.

[53] J. Brownlee, "Tune hyperparameters for classification machine learning algorithms." [En línea]. Disponible en: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

[54] E. Lutins, "Grid searching in machine learning: Quick explanation and python implementation." [En línea]. Disponible en: https://medium.com/@elutins/grid-searching-in-machine-learning-quick-explanation-and-python-implementation-550552200596

[55] Spark, "Documentation spark." [En línea]. Disponible en: https://spark.apache.org/docs/2.2.0/api/python/pyspark.mllib.html#pyspark.mllib.evaluation.BinaryClassificationMetrics

[56] Scikit-Learn, "Documentation scikit-learn." [En línea]. Disponible en: https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

[57] Databricks, "Apache spark mllib pipelines and structured streaming example." [En línea]. Disponible en: https://docs.databricks.com/applications/machine-learning/mllib/mllib-pipelines-and-stuctured-streaming.html